Escola Politècnica Superior
d'Edificació de Barcelona
UNIVERSITAT POLITÈCNICA DE CATALUNYA

Ingeniería Técnica Topográfica

Proyecto final de carrera

Modelado en tres dimensiones con luz estructurada

Proyectista: Viladevall Martinez, Jaume

Directores: Buill Pozuelo, Felipe

Prades Valls, Albert

Convocatoria: Abril 2014

RESUMEN

Este trabajo tiene como objetivo crear un modelo en tres dimensiones de un objeto a partir de una técnica conocida como luz estructurada. El sistema, montado sobre un trípode, está compuesto por un proyector y dos cámaras monocromas situadas sobre una barra de aluminio que capturarán las imágenes proyectadas sobre una figura desde dos puntos de vista.

Una vez las imágenes estén corregidas de errores sistemáticos y normalizadas (epipolarizadas) se podrán medir las coordenadas imagen de puntos correspondientes en ambos fotogramas.

A partir de esta información podremos encontrar la disparidad entre imágenes y su profundidad creando una nube de puntos con coordenadas conocidas en el espacio.

ÍNDICE

1 INTRODUCCIÓN

Hoy en día existen diferentes técnicas de obtención de coordenadas de nubes de puntos de alta calidad y de forma instantánea, estos escáneres utilizan luz láser para trabajar con una alta precisión o luz infrarroja para eliminar problemas causados por la luz ambiental.

El objetivo de este proyecto es crear un sistema fotogramétrico capaz de crear una nube de puntos suficientemente buena para efectuar un escaneado de figuras, objetos o caras humanas, con errores tolerables y que a la vez sea un sistema que tenga un coste reducido.

Por último cabe comentar que el escaneado obtenido de este trabajo tiene aplicaciones en muchos ámbitos como la medicina, odontología, estudios de superficies y en un futuro en la digitalización de objetos creando planos para impresión en 3D.

2 NÚCLEO DE LA MEMORIA

2.1 VISIÓN EN TRES DIMENSIONES

2.1.1 Visión estereoscópica

La visión estereoscópica es la percepción de profundidad a partir de imágenes en dos dimensiones. Nuestros ojos, siendo un mecanismo de visión estéreo, son capaces de apreciar diferentes distancias y volúmenes en el entorno. Debido a su separación, que suele estar entre 45mm y 75mm siendo 65mm la más habitual, los ojos obtienen dos imágenes con pequeñas diferencias entre ellas; estas diferencias son conocidas como disparidad.

Nuestro cerebro procesa la disparidad entre las imágenes y finalmente percibimos nuestro entorno con profundidad.

2.1.1.1 Visión monocular

La visión monocular es cuando obtenemos imágenes a partir de un único ojo. Este tipo de visión nos da una percepción limitada de la profundidad ya que sólo vemos desde una dirección y recibimos información bidimensional.

2.1.1.2 Visión binocular

La visión binocular, que proviene del latín “bini” doble y “oculus” ojo, es cuando nuestro cerebro recibe una imagen de ambos ojos. (fig.1)

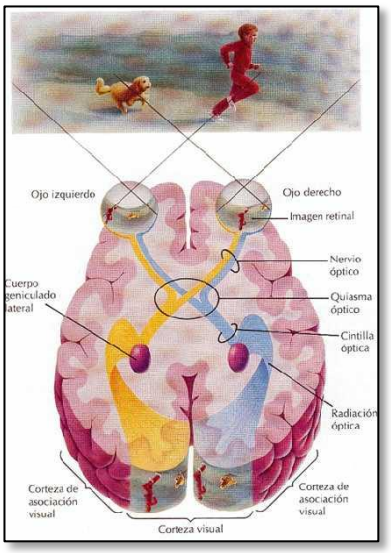


Fig. 1. Cortesía de www.trialx.com

2.2 FUNDAMENTOS FOTOGRAMÉTRICOS

En primer lugar se explican los procesos fotogramétricos que se han implementado en el código para hallar la nube de puntos del objeto o figura escaneada.

2.2.1 Distorsión de la lente

Por muy buena que sea, cualquier objetivo del mercado tiene algún tipo de deformación, estas deformaciones en las lentes crean aberraciones que desvían el sistema óptico de un comportamiento ideal, esto imposibilita reproducir matemáticamente la imagen de cualquier objeto. La aberración geométrica más importante que causa el desplazamiento de los puntos que forman la imagen respecto de su posición ideal se llama distorsión. Esta tiene componentes radiales y tangenciales.

2.2.1.1 Distorsión radial

La distorsión radial es simétrica respecto al eje óptico y crea un desplazamiento del punto imagen hacia afuera o hacia dentro respecto del punto principal de simetría. Este tipo de distorsión aparece por la esfericidad de la lente y se suele tomar como modelo matemático la siguiente expresión:

$$\Delta r = K_1 r^3 + K_2 r^5 + K_3 r^7 + \cdots$$

$$\Delta x_r = x' \frac{\Delta r}{r'}$$

$$\Delta y_r = y' \frac{\Delta r}{r'}$$

Ec. 1

Siendo:

$$r = \sqrt{(x - x_0)^2 + (y - y_0)^2}$$

Ec. 2

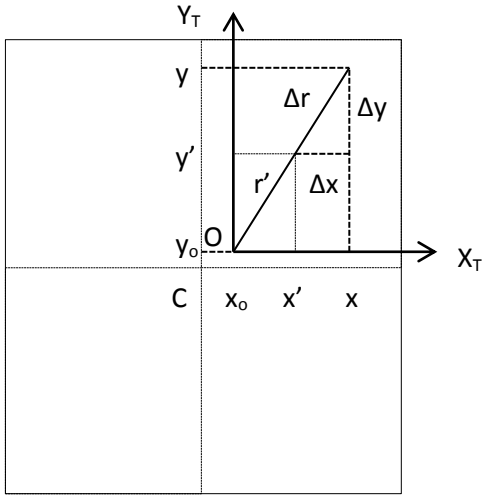


Fig. 2

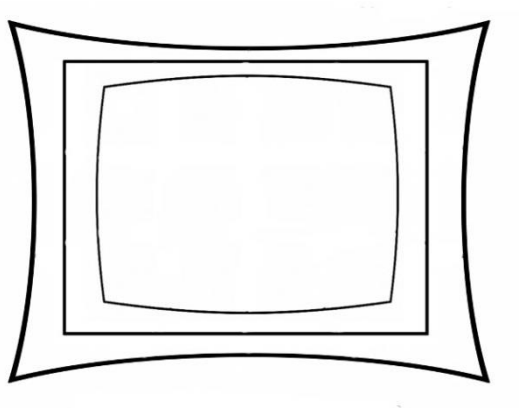


Fig. 3

En la figura 3 vemos los tres ejemplos de distorsión más comunes: distorsión en corsé siendo la más exterior, distorsión en barrilete siendo la del interior y la del medio sería sin distorsión o nula.

Teniendo en cuenta la figura 2, las coordenadas imagen corregidas se obtienen con las siguientes expresiones:

$$x' = (x - x_0) - \Delta x = (x - x_0)\left(1 - \frac{\Delta r}{r}\right) = (x - x_0)(1 - K_1 r^2 - K_2 r^4 - K_3 r^6 - \cdots)$$

$$y' = (y - y_0) - \Delta y = (y - y_0)\left(1 - \frac{\Delta r}{r}\right) = (y - y_0)(1 - K_1 r^2 - K_2 r^4 - K_3 r^6 - \cdots)$$

Ec. 3

A continuación se aprecia la distorsión en barrilete que tienen las lentes que usamos. Se observa como los puntos que deberían seguir la línea amarilla se desplazan hacia el exterior en los centros.

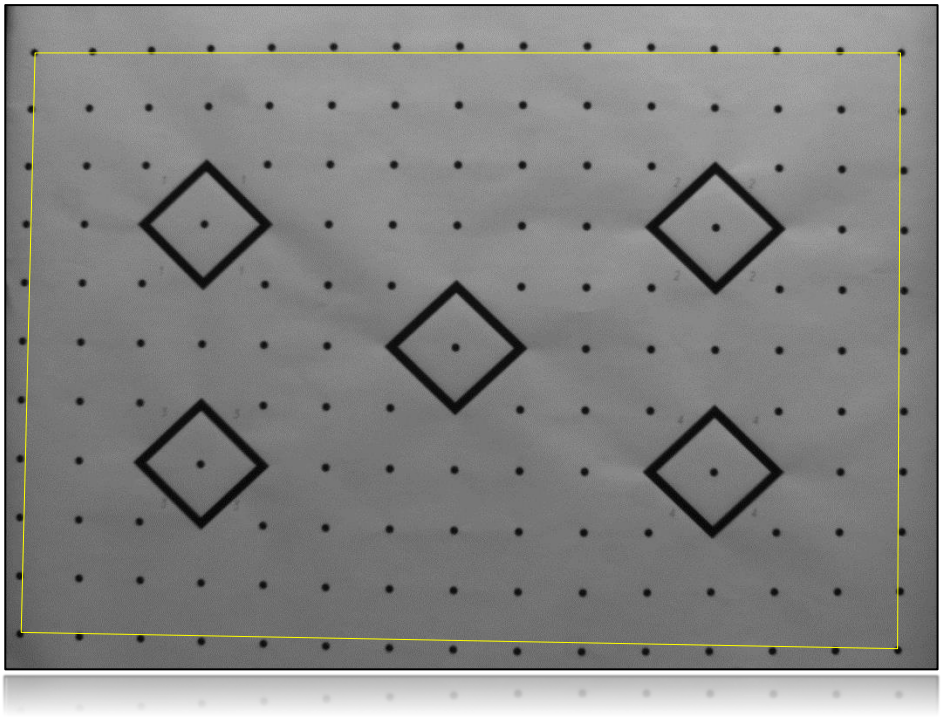


Fig. 4. Plantilla de calibración con distorsión en barrilete

2.2.1.2 Distorsión tangencial

La distorsión tangencial se produce debido a que el punto principal de la lente esta descentrado. Al igual que la radial se desplaza el punto imagen ideal pero en una dirección ortogonal a la distorsión radial.

$$\Delta x_t = P_1(r'^2 + 2x'^2) + 2P_2x'y'$$

$$\Delta x_t = P_1(r'^2 + 2x'^2) + 2P_2x'y'$$

Ec. 4

Siendo P_1 y P_2 los parámetros tangenciales, si juntamos las ecuaciones de la distorsión radial y tangencial obtenemos la corrección conjunto:

$$x = x' + x'\frac{\Delta r}{r'} + P_1(r'^2 + 2x'^2) + 2P_2x'y'$$

$$y = y' + y'\frac{\Delta r}{r'} + P_2(r'^2 + 2y'^2) + 2P_1x'y'$$

Ec. 5

2.2.2 Puntos homólogos

Para obtener los parámetros de orientación relativa es de vital importancia hallar los puntos homólogos en las imágenes de ambas cámaras.

Ya que hay que relacionar los puntos vistos con una cámara con los vistos con la otra. Para ello se ha creado un patrón de puntos para proyectar. (Fig. 5)

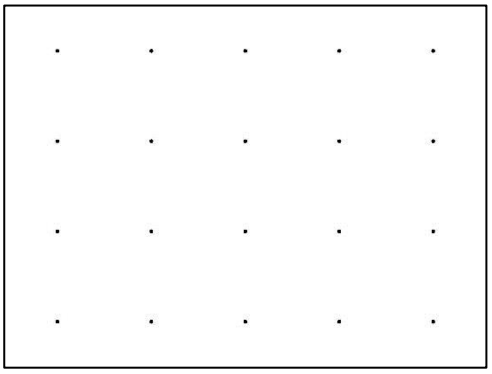


Fig. 5. Patrón de puntos

Las cámaras luego capturan este patrón:

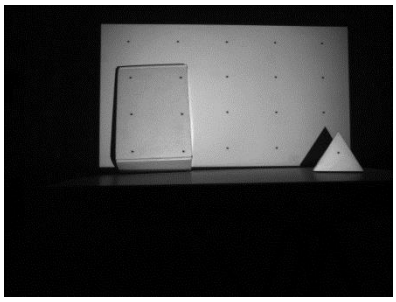


Fig. 6. Patrón capturado por la cámara izquierda

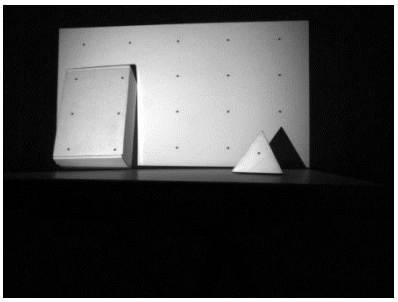


Fig. 7. Patrón capturado por la cámara derecha

Después de corregir las imágenes de distorsión radial y tangencial, se ha creado un programa que busca los puntos proyectados y calcula sus coordenadas:

```
%Encontrar los puntos y coordenadas.
I=imread('der_C.jpg');
[c, r] = imfindcircles(I,[4 6], ...
    'ObjectPolarity','dark','Sensitivity',0.92,'Method','twostage');
%Mostrar círculos.
imshow(I);viscircles(c,r);
```

Como se puede observar en la imagen de la derecha el programa busca puntos por toda la imagen con las características definidas y dibuja círculos para que podamos visualizar que ha seleccionado los puntos adecuados. Además nos da las coordenadas y el radio de los puntos seleccionados.

Una vez tenemos los puntos y sus coordenadas con ambas cámaras podremos aplicar estos puntos homólogos para hallar los parámetros de giro y traslación mediante el proceso de orientación relativa.

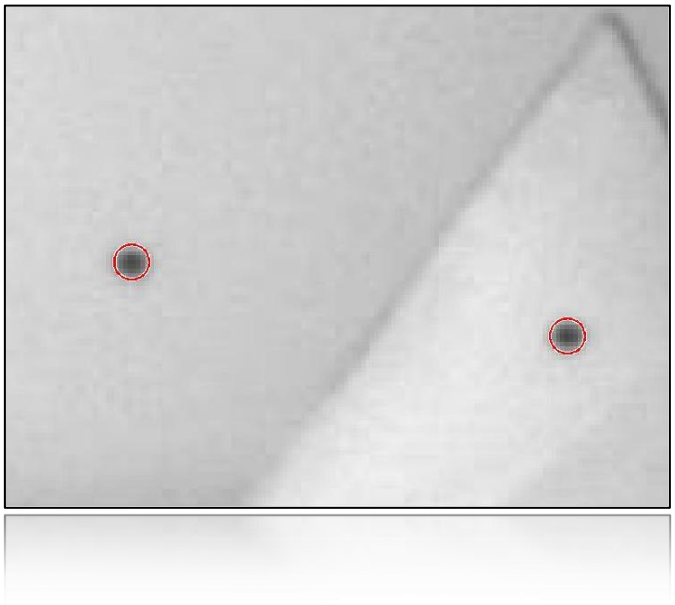


Fig. 8. Puntos homólogos seleccionados.

2.2.3 Calibración de la cámara

Para poder corregir la distorsión primero hay que obtener los parámetros internos de la cámara, concretamente los parámetros de distorsión radial y tangencial, en este caso se ha usado el programa de *TOPCON*: *Image Master Calibration*. Para ello se han tomado cinco imágenes con las dos cámaras de un patrón de calibración proporcionado por el mismo programa.

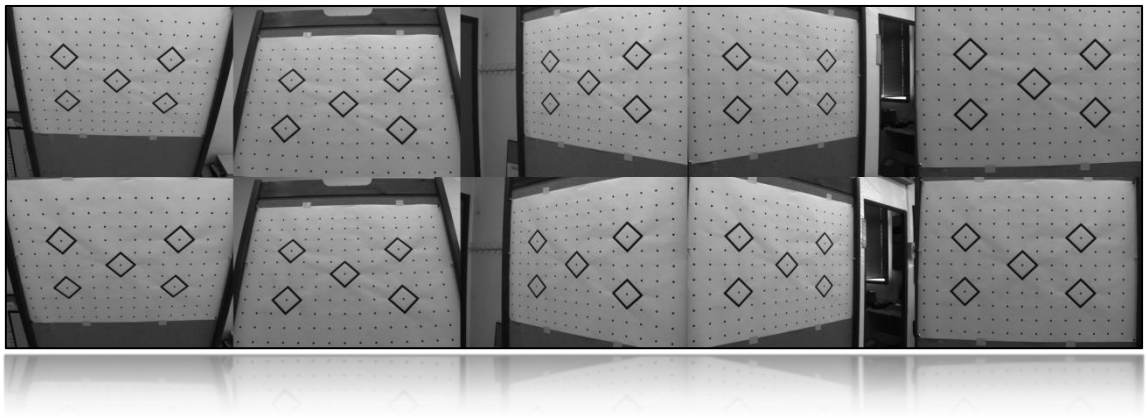


Fig. 9. Fotografías de la plantilla de calibración proporcionada por *Image Master.*

Una vez cargamos las imágenes, se abren de una en una y seleccionamos los puntos situados dentro del rombo en cada imagen:

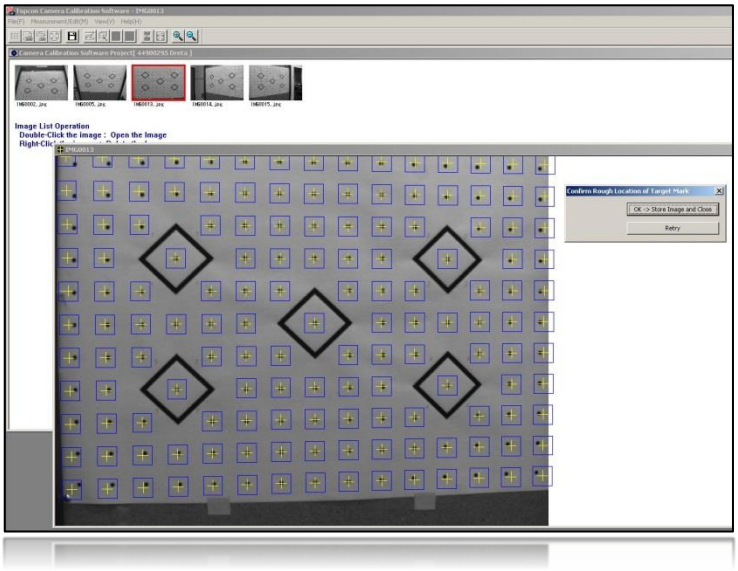


Fig. 10. Puntos seleccionados de forma automática por el programa.

Finalmente el programa hará el cálculo de los parámetros de distorsión y puntos principal de la lente para que podamos corregir las imágenes.

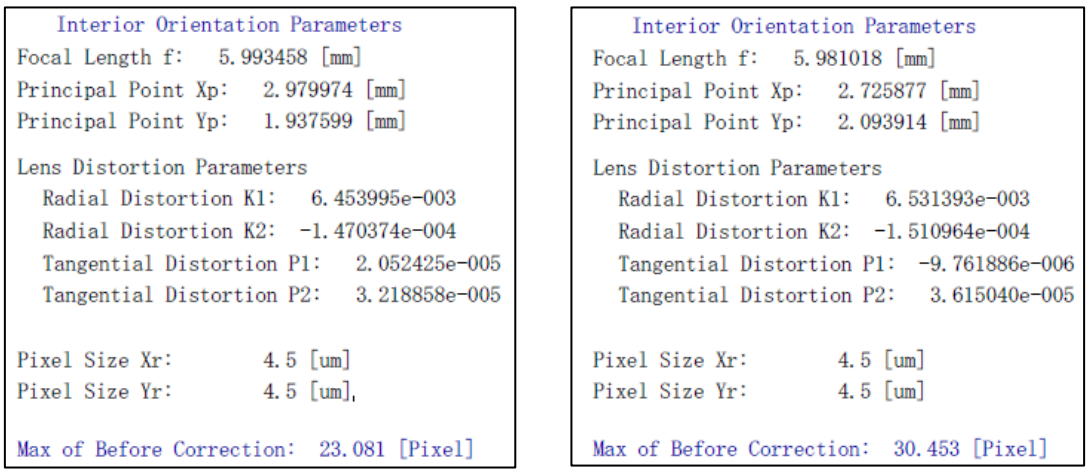


Fig. 11. Parámetros de orientación de cámara izquierda y derecha respectivamente

Una vez tenemos estos parámetros que son únicos para cada cámara y objetivo (Fig.11), se ha creado un código en *MATLAB* que corrige las imágenes según sus parámetros de orientación interna.

2.2.4 Orientación relativa

La orientación relativa es un proceso que permite determinar las orientaciones angulares y posiciones relativas que se dieron en el momento de toma de dos pares de imágenes. Se basa en la correspondencia entre puntos homólogos en un par de imágenes fotográficas. Los puntos homólogos de dos imágenes estéreo tomadas al mismo instante tendrá un desplazamiento y un giro una respecto a otra, ese giro y desplazamiento se puede cuantificar en parámetros.

La orientación relativa se puede llevar a cabo mediante dos condiciones, la de coplanaridad o colinealidad. En este caso se ha utilizado la condición de coplanaridad mediante rotaciones. Para ello se han fijado cinco de los doce parámetros, seis de rotación y seis de traslación.

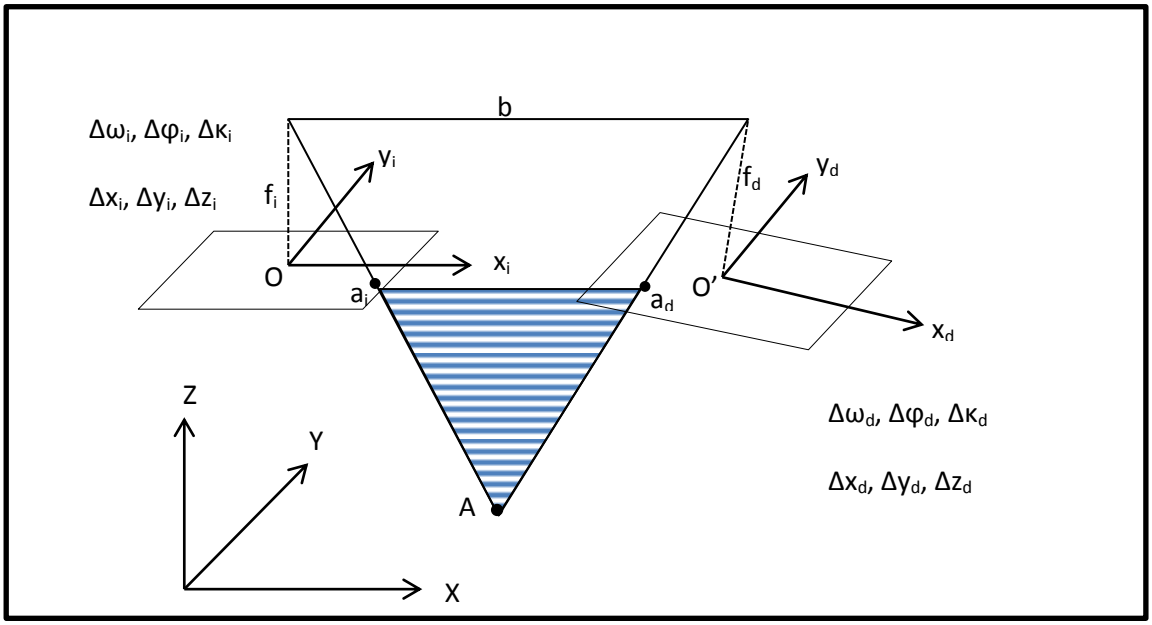


Fig. 12

La coplanaridad impone que los rayos ópticos que contienen los puntos homólogos, se cortan y forman un plano:

$$aX + bY + cZ + d = 0$$

Ec. 6

La expresión matricial de la condición de coplanaridad viene definida por:

$$F = \begin{vmatrix} b_x & b_y & b_z \\ x_i' & y_i' & z_i' \\ x_d' & y_d' & z_d' \end{vmatrix} = 0$$

Ec. 7

Siendo:

-$b_x, b_{y,} b_z$ las bases del modelo estereoscópico.

-$x_i', y_i'\ z_i'$ y $x_d', y_d'\ z_d'$ las coordenadas imagen del fotograma izquierdo y derecho respectivamente en un sistema de coordenadas paralelo al tridimensional, es decir:

$$\begin{pmatrix} x_i' \\ y_i' \\ z_i' \end{pmatrix} = R^i_{\omega\varphi\kappa} \begin{pmatrix} x_i \\ y_i \\ f_i \end{pmatrix} \qquad \begin{pmatrix} x_d' \\ y_d' \\ z_d' \end{pmatrix} = R^d_{\omega\varphi\kappa} \begin{pmatrix} x_d \\ y_d \\ f_d \end{pmatrix}$$

Ec. 8

Siendo:

-f_i y f_d las focales de las cámaras izquierdas y derechas respectivamente

-Matriz de rotación:

$$R = \begin{pmatrix} \cos\varphi\cos\kappa & -\cos\omega\sin\kappa + \sin\omega\sin\varphi\cos\kappa & \sin\omega\sin\kappa + \cos\omega\sin\varphi\cos\kappa \\ \cos\varphi\sin\kappa & \cos\omega\cos\kappa + \sin\omega\sin\varphi\sin\kappa & -\sin\omega\cos\kappa + \cos\omega\sin\varphi\sin\kappa \\ -\sin\varphi & \sin\omega\cos\varphi & \cos\omega\cos\varphi \end{pmatrix}$$

Ec. 9

El desarrollo del determinante da un sistema de ecuaciones no lineales. Por lo tanto, el primer paso es quedarnos con la parte lineal del desarrollo en serie de Taylor de la función *F=0*, despreciando infinitésimos de segundo orden y considerando nulos los errores de las coordenadas imagen tenemos:

$$(F)_0 + \left(\frac{\partial F}{\partial\varphi_i}\frac{\partial F}{\partial\kappa_i}\frac{\partial F}{\partial\omega_d}\frac{\partial F}{\partial\varphi_d}\frac{\partial F}{\partial\kappa_d}\right)_0 \begin{pmatrix} d\varphi_i \\ d\kappa_i \\ d\omega_d \\ d\varphi_d \\ d\kappa_d \end{pmatrix} = 0$$

$$(F)_0 + A_{11}d\varphi_i + A_{12}d\kappa_i + A_{13}d\omega_d + A_{14}d\varphi_d + A_{15}d\kappa_d = 0$$

Ec. 10

Siendo:

$$A_{11} = \begin{vmatrix} b_x & b_y & b_z \\ \frac{\partial x_i'}{\partial\varphi_i} & \frac{\partial y_i'}{\partial\varphi_i} & \frac{\partial z_i'}{\partial\varphi_i} \\ x_d' & y_d' & z_d' \end{vmatrix}_0 \qquad A_{12} = \begin{vmatrix} b_x & b_y & b_z \\ \frac{\partial x_i'}{\partial\kappa_i} & \frac{\partial y_i'}{\partial\kappa_i} & \frac{\partial z_i'}{\partial\kappa_i} \\ x_d' & y_d' & z_d' \end{vmatrix}_0 \qquad A_{13} = \begin{vmatrix} b_x & b_y & b_z \\ x_i' & y_i' & z_i' \\ \frac{\partial x_d'}{\partial\omega_d} & \frac{\partial y_d'}{\partial\omega_d} & \frac{\partial z_d'}{\partial\omega_d} \end{vmatrix}_0$$

$$A_{14} = \begin{vmatrix} b_x & b_y & b_z \\ x_i' & y_i' & z_i' \\ \frac{\partial x_d'}{\partial\varphi_d} & \frac{\partial y_d'}{\partial\varphi_d} & \frac{\partial z_d'}{\partial\varphi_d} \end{vmatrix}_0 \qquad A_{15} = \begin{vmatrix} b_x & b_y & b_z \\ x_i' & y_i' & z_i' \\ \frac{\partial x_d'}{\partial\kappa_d} & \frac{\partial y_d'}{\partial\kappa_d} & \frac{\partial z_d'}{\partial\kappa_d} \end{vmatrix}_0$$

Ec. 11

La matriz de diseño (A) estará compuesta de tantas ecuaciones como puntos homólogos y de cinco incógnitas siendo ϕ_i ,κ_i, ω_d, ϕ_d, κ_d habiendo fijado ω_i, b_y y b_z como cero y b_x como la distancia entre cámaras.

Calcularemos mediante un ajuste de mínimos cuadrados y de forma iterativa los parámetros diferenciales de los giros de la siguiente forma:

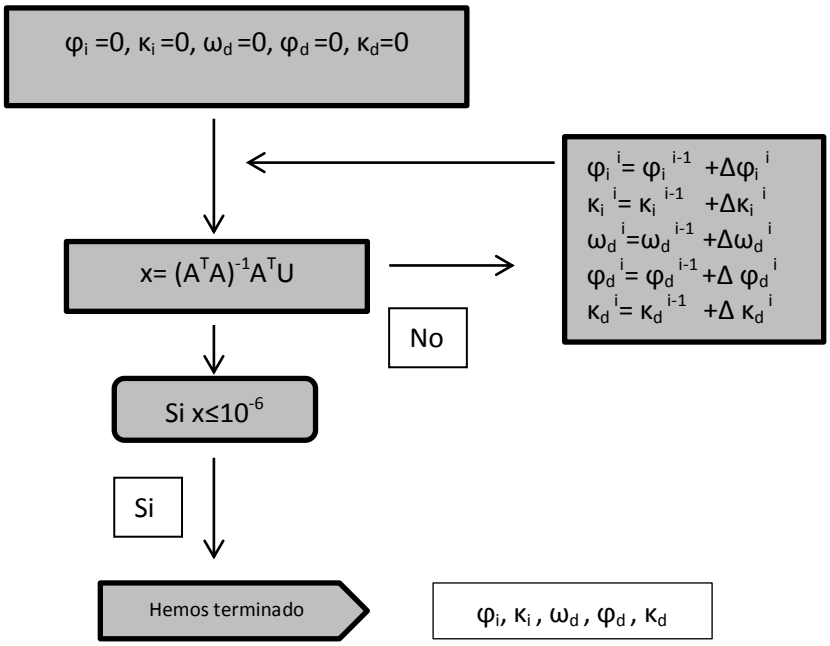


Fig. 13

2.2.5 Correspondencia de imagen

La correspondencia de imagen, también llamada correlación de imagen es el proceso de encontrar detalles o entidades homologas lineales y puntuales en una o varias imágenes. En el caso de este proyecto se trata de encontrar el homólogo de cada punto detectado en la franja de luz de todas las imágenes tomadas. Una vez tenemos todas coordenadas de los puntos de la cámara izquierda junto a su homólogo en la cámara derecha, encontramos fácilmente las coordenadas modelo de nuestro objeto.

De los tres métodos generales de correspondencia de imagen, se usará el método de correspondencia basada en intensidades *'intensity-based matching (IBM)'.* Concretamente es el conocido como el procedimiento *correlación mínimo cuadrática*.

La geometría epipolar de una toma estereoscópica nos dice que un punto en la imagen de la izquierda, x_i, encuentra su homólogo en la imagen de la derecha, x_d, siguiendo su correspondiente línea epipolar. Esta línea epipolar se determina a partir de la intersección del plano epipolar a las respectivas imágenes. (Fig. 14)

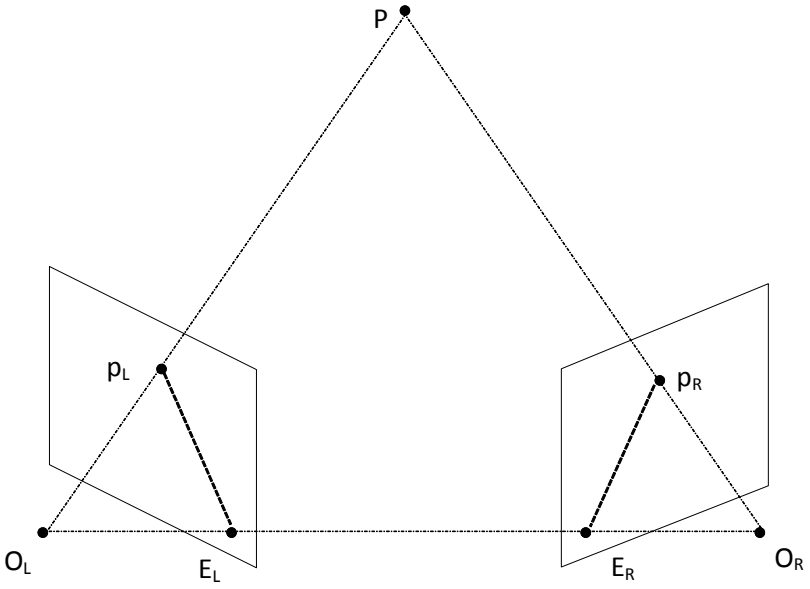


Fig. 14

De esta forma la búsqueda de un punto homólogo es tan simple como un espacio bidimensional a un espacio unidimensional. Si las tomas fotográficas son normalizadas los puntos homólogos no presentaran paralaje y la correspondencia de las líneas epipolares será en dirección horizontal.

En la siguiente figura se aprecian las 16 posibles correspondencias que se pueden realizar haciendo la intersección de las rectas I_1, I_2, I_3 y I_4 con D_1, D_2, D_3 y D_4 de los cuatro puntos alineados de las dos imágenes. Si observamos la figura veremos que sólo cuatro de estas intersecciones son correctas, concretamente las que son representadas por el punto negro. Las otras correspondencias, representadas por un cuadro, dan un desplazamiento y profundidad falsa. Estas últimas se darían en el caso de que hiciéramos la correspondencia de imágenes diferentes entre las cámaras.

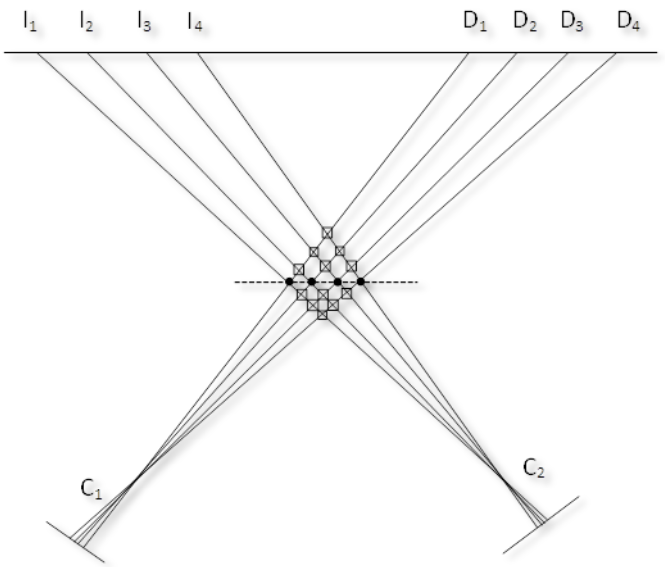


Fig. 15

Para evitar una situación de falsa correspondencia, la sincronización entre la toma fotográfica y la proyección de patrón ha sido una parte importante del proyecto que ha necesitado varios retoques para poder garantizar una buena sincronización sin perjudicar mucho el tiempo de trabajo.

2.2.5.1 Sincronización

La sincronización tiene un peso importante a la hora de poder hacer una correspondencia adecuada de la toma fotográfica. Sin una sincronización correcta la está nos da unas profundidades falsas que convierten algo como un plano en una figura escalonada con diferentes profundidades. La figura 15 nos muestra un caso teórico de una correspondencia correcta e incorrecta.

A continuación tenemos un caso de un plano representado con una pared con correspondencia falsa debido a una mala sincronización entre la toma fotográfica y la proyección del patrón de luz.

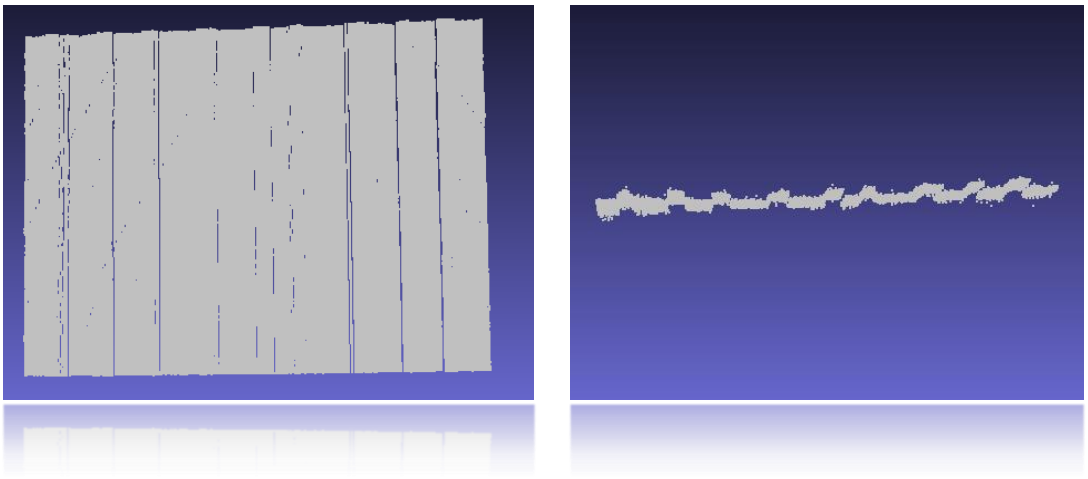


Fig. 16. Vista en alzado y planta de una pared con una correspondencia errónea

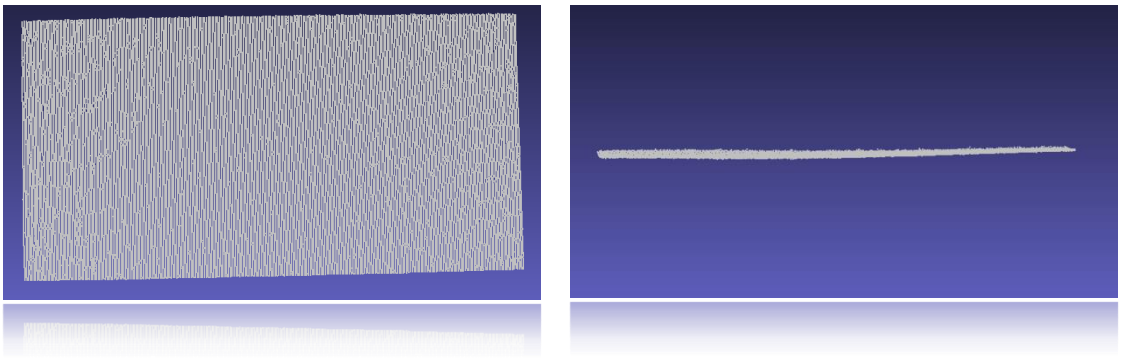


Fig. 17. Vista en alzado y planta de una pared con una correspondencia correcta

2.5.5.2 Correspondencia mínimo cuadrática con constreñimientos geométricos

Esta correspondencia permite determinar las entidades conjugadas y las coordenadas en el espacio del modelo. Para este método es necesario conocer los parámetros de orientación de las imágenes fotográficas.

Concretamente se ha usado el constreñimiento de colinealidad, este puede utilizarse cuando la formación de la imagen sigue la ley de proyección perspectiva. Las posiciones de puntos objeto sobre las imágenes son determinadas por las ecuaciones de colinealidad.

$$\bar{x} = x - x_0 = -f\frac{m_{11}(X - X_L) + m_{12}(Y - Y_L) + m_{13}(Z - Z_L)}{m_{31}(X - X_L) + m_{32}(Y - Y_L) + m_{33}(Z - Z_L)} = -F^x$$

$$\bar{y} = y - y_0 = -f\frac{m_{21}(X - X_L) + m_{22}(Y - Y_L) + m_{23}(Z - Z_L)}{m_{31}(X - X_L) + m_{32}(Y - Y_L) + m_{33}(Z - Z_L)} = -F^y$$

Ec. 12

```
for i=1:L %Ecuación colinealidad imagen izquierda
    xpi(i) = -f*(left(i,1)*RI(1,1) + left(i,2)*RI(1,2) - f*RI(1,3))...
        /(left(i,1)*RI(3,1) + left(i,2)*RI(3,2) - f*RI(3,3));
    ypi(i) = -f*(left(i,1)*RI(2,1) + left(i,2)*RI(2,2) - f*RI(2,3))...
        /(left(i,1)*RI(3,1) + left(i,2)*RI(3,2) - f*RI(3,3));
end
for i=1:R %Ecuación colinealidad imagen derecha
     xpd(i) = -f*(right(i,1)*RD(1,1) + right(i,2)*RD(1,2) - f*RD(1,3))...
         /(right(i,1)*RD(3,1) + right(i,2)*RD(3,2) - f*RD(3,3));
     ypd(i) = -f*(right(i,1)*RD(2,1) + right(i,2)*RD(2,2) - f*RD(2,3))...
         /(right(i,1)*RD(3,1) + right(i,2)*RD(3,2) - f*RD(3,3));
end
```

2.2.6 Obtención de coordenadas modelo

Una vez tenemos los parámetros de giro y traslación tenemos que aplicar estos a los puntos que nos interesen, en nuestro caso como solamente estamos trabajando con las coordenadas de la franja que se desplaza, aplicaremos estos parámetros a las coordenadas de todos los puntos de cada franja.

La intersección de rayos homólogos proporciona la posición de un punto m sobre el modelo. La definición de ambas rectas en el espacio se expresa de la siguiente forma:

$$\frac{x_{m-}x_L}{x_i'} = \frac{y_{m-}y_L}{y_i'} = \frac{z_{m-}z_L}{z_i'} = N$$

$$\frac{x_{m-}x_L'}{x_d'} = \frac{y_{m-}y_L'}{y_d'} = \frac{z_{m-}z_L'}{z_d'} = M$$

Ec. 13

Las ecuaciones contienen implícitamente a las componentes de la base:

$$b_x = x_{L'} - x_L$$
$$b_y = y_{L'} - y_L$$
$$b_z = z_{L'} - z_L$$

Ec. 14

Se determinan los factores de escala según:

$$M = \frac{b_x z_i' - b_z x_i'}{x_i' z_d' - z_i' x_d'} \qquad N = \frac{b_x x_d' - b_z z_d'}{x_i' z_d' - z_i' x_d'}$$

Ec. 15

A continuación se muestra como a partir de estas ecuaciones se pueden calcular las coordenadas modelo.

$$x_m = N \cdot x_i' + x_L$$
$$y_m = \frac{1}{2}(y_L + y_L' + y_i' N + y_d' M) \quad \begin{cases} y_m = N \cdot y_i' + y_L \\ y_m = M \cdot y_d' + y_L' \end{cases}$$
$$z_m = N \cdot z_i' + z_L$$

Ec. 16

2.3 INSTRUMENTACIÓN

2.3.1 Pruebas

A lo largo de este proyecto se han hecho pruebas con varias cámaras y proyectores. Inicialmente se empezó utilizando una *Nikon D70* con una lente *Sigma* de 30mm de focal.

Fig. 18*. Nikon D70* y *Sigma* 30mm cortesía de *Nikon* y *Sigma*

Con esta cámara el sistema en lugar de tener dos cámaras sólo tenía una, y el proyector actuaba de segunda cámara en cuanto a la correspondencia entre imágenes. El proyector en cuestión era un *BenQ MP610.*

Este proyector al no ser de lámpara LED, podía perjudicar la toma de datos ya que las lentes se podrían dilatar debido al calor y algunas imágenes mostrarían distorsiones diferentes de otras. Esta y el tamaño del proyector fueron las razones por las que se optó cambiar a un pico proyector.

Fig. 19. Cortesía de *BenQ*

Una vez se decidió pasar a un proyector más pequeño de lámpara LED, se planteó trabajar con dos cámaras en lugar de una, de esta forma la correspondencia entre imágenes se podría hacer de manera más simple. Para ello se empezó a hacer pruebas con una *Canon 450D* con una lente *Sigma aspherical* de 24mm de focal. Pero el problema principal que tenía usar dos cámaras *Canon 450D* era que no se podía capturar las imágenes de forma rápida, en el caso de que tuvieran función de video se podría capturar un video del patrón y separarlo por fotogramas, pero con estas cámaras en concreto sólo se podía tomar imágenes con el programa propio de *Canon*.

Con este programa la sincronización resulto difícil y para poder efectuar todas las tomas fotográficas de los patrones de la izquierda y derecha se necesitaba aproximadamente 1 hora y media, unos 45 minutos por cámara.

Fig. 20. *Canon 450D* y lente *sigma aspherical* 24mm cortesía de *Canon* y *Sigma*.

2.3.2 Sistema final

El sistema final que se ha usado consta de dos cámaras, un proyector y un trípode. Conectado a un ordenador este conjunto de objetos se convierte en un sistema capaz de capturar un objeto o modelo y transformarlo en una nube de puntos con coordenadas conocidas en el espacio.

Las cámaras usadas en este caso son dos cámaras CCD monocroma de *Sony* con conexión USB 2.0 de la casa *Imaging Source*. Concretamente el modelo *DMK 41BU02*. Con objetivos de 6mm de focal de la marca *Pentax*.

Fig. 21. Imaging Source DMK 41BU02

Estas cámaras llevan un sensor CCD ½" *Sony* modelo *ICX205AL*, presente en muchas cámaras astronómicas. Tiene un tamaño de píxel de 4.65µm y captura unas imágenes de 1280x960 de resolución.

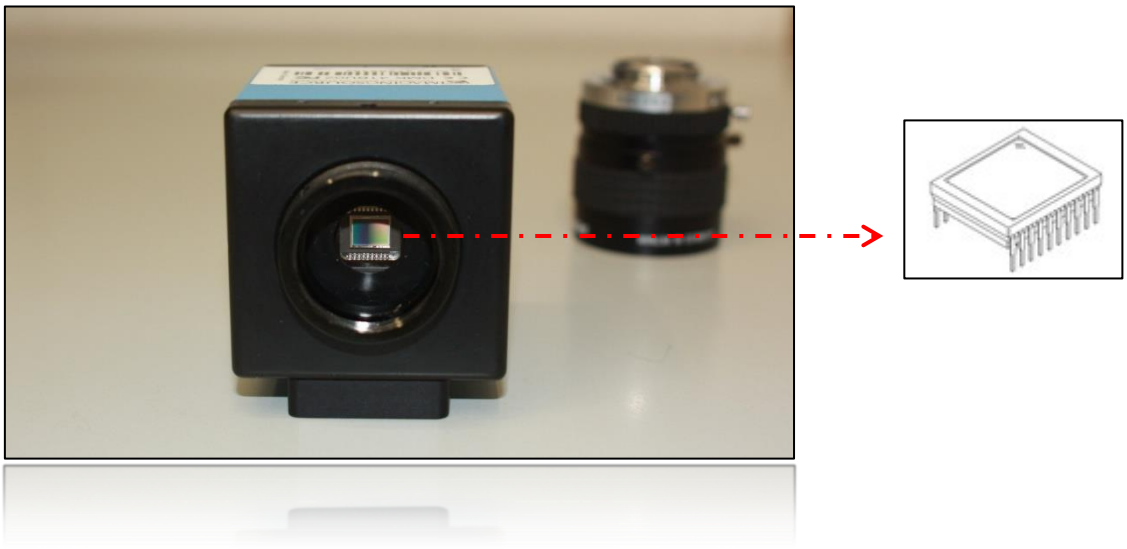


Fig. 22. Vista del sensor CCD ½" de *Sony ICX205AL*

El proyector empleado es un *Acer C120*. Es un pico proyector con tecnología *DLP* y una lámpara LED. El *DLP* o "Procesamiento Digital de Luz" está desarrollado por *Texas Instrument.* El tamaño reducido del proyector ha sido fundamental para el desarrollo del sistema.

Fig. 23. Pico proyector LED *Acer c120*

En los proyectores *DLP* la imagen es creada por un conjunto de espejos dispuestos en una matriz sobre un chip semiconductor llamado *Digital Micromirror Device*. Cada píxel en la imagen proyectada es representado por un espejo, estos se inclinan rápidamente para reflejar luz a la pantalla dándole color al píxel o no reflejando haciendo que el píxel sea negro.

El sistema está sujeto por una barra de aluminio de un metro de longitud donde las dos cámaras y el proyector van anclados a una guía que permite desplazar las cámaras modificando su distancia base obteniendo diferentes resultados.

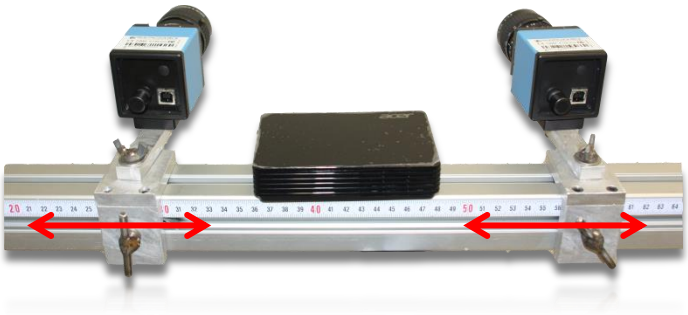


Fig. 24. Movimiento horizontal que permite modificar la base

La barra además tiene una cinta métrica en la parte trasera que nos facilita la distancia base aproximada.

Fig. 25. Barra de aluminio

El sistema de anclaje también permite modificar el ángulo entre las cámaras:

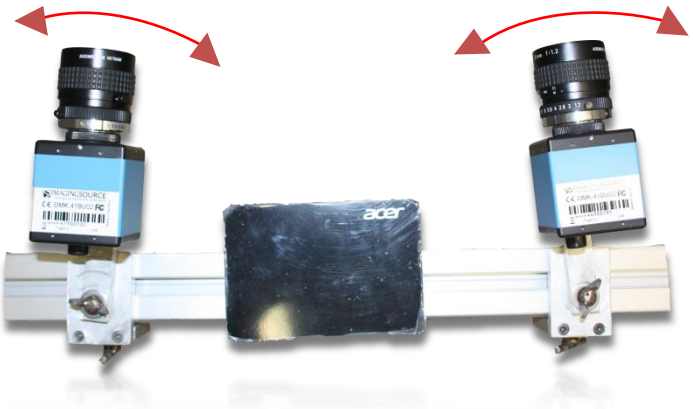
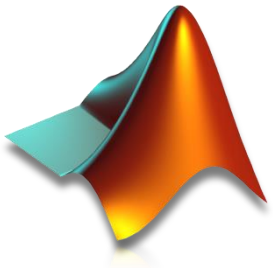


Fig. 26. Movimiento para modificar el ángulo de las cámaras

2.4 EL PROCESO

Este apartado estará centrado en la parte práctica del trabajo, primero se hablará sobre los programas usados y finalmente el paso a paso para obtener la nube de puntos del modelo en tres dimensiones.

2.4.1 *MATLAB* y otros programas



MATLAB, que es una abreviatura de *MATrix LABoratory* o laboratorio de matrices, es un programa y herramienta matemática con un lenguaje propio conocido como *M*. Es un programa muy usado en centros de investigación y desarrollo con un potente procesador de imágenes y de matrices, ideal para un trabajo como este.

Ante todo para poder llevar a cabo este proyecto he tenido que aprender y familiarizarme con el lenguaje de programación de *MATLAB* y con el programa en si ya que era la primera vez que lo usaba y mis conocimientos de programación eran muy básicos. Las razones por las que se decidió usar este programa fueron por su potente procesador de imagen y por ser más fácil de aprender y usar que otros lenguajes de programación.

MATLAB permite manipular imágenes como matrices, cada píxel corresponde a una coordenada de la matriz que representa la imagen. También cabe destacar que *MATLAB* contiene ya muchas funciones incorporadas en su código como la operación mínimo cuadrática.

```
%Mínimos cuadrados
dx=A\U';
V=A*dx-U';
```

Otra ventaja que se encontró con *MATLAB* es que *Imaging Source,* quien fabrica las cámaras empleadas, tiene una extensión llamada *IC Matlab Plugin for Matlab R2013b*, concretamente para la caja de herramientas de *MATLAB: Image Acquisition Toolbox*, que permite capturar imágenes y video directamente de cámaras detectadas por el sistema. De este modo el paso previo de capturar las imágenes mediante otro programa para luego cargarlas con *MATLAB* se ha incorporado directamente en el código.

Desde *Image Acquisition Toolbox*, se puede modificar parámetros de la cámara como la abertura, el contraste, el parámetro gamma, etc.

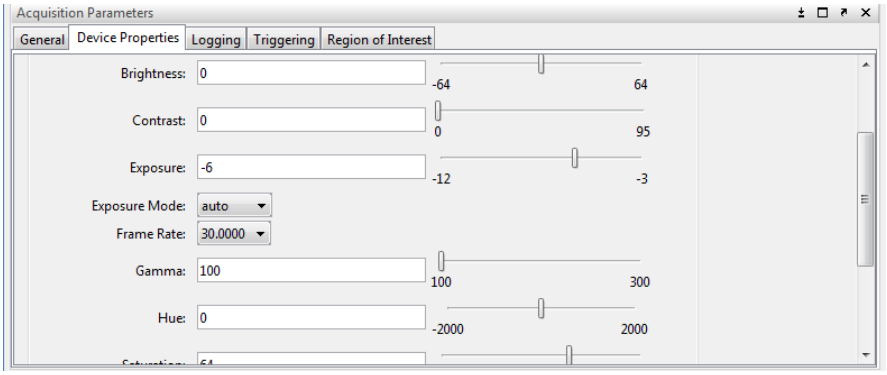


Fig. 27. Pantalla para modificar parámetros de la cámara

Estos parámetros se pueden cambiar también en el propio programa que creamos modificándolos en cualquier momento o línea del código:

```
%Identificamos las dos cámaras y sus propiedades para poder
modificarlas.
obj = videoinput('winvideo', 2);
obj2= videoinput('winvideo', 3);
src = getselectedsource(obj);
src2= getselectedsource(obj2);

%Modificamos gain, gamma y exposición de las dos cámaras.
src.Gain=400;
src.Gamma=110;
src.Exposure = -4;
src2.Gain=400;
src2.Gamma=110;
src2.Exposure = -4;
```

También cabe destacar que hay muchos programas ya hechos y colgados por usuarios en internet que pueden ser de utilidad. Cuando se hace la proyección de la imagen del patrón para ser fotografiada, se efectúa con el propio *MATLAB* de forma que crea y muestra el patrón a la vez, optimizando tiempo y espacio. Como *MATLAB* no permite proyectar una imagen en pantalla completa porque no está programado de así, al proyectar el patrón mostraba la imagen en una ventana de figura. (fig. 28)

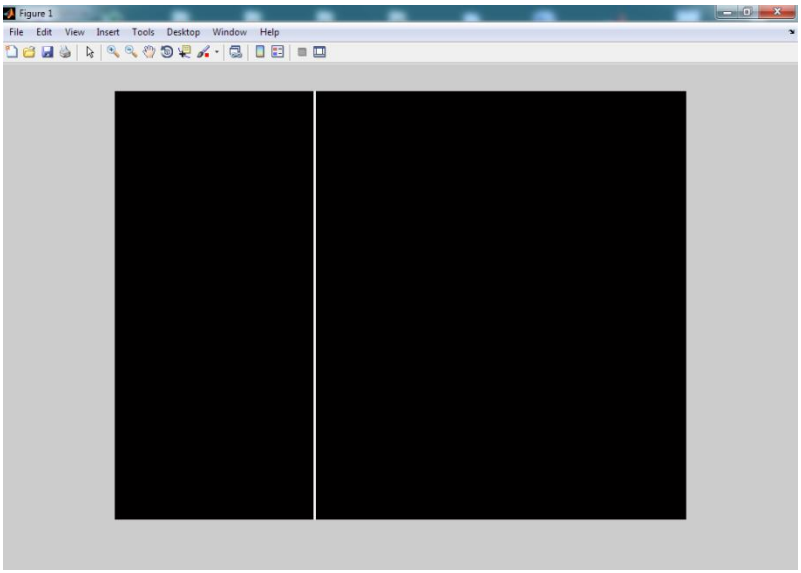


Fig. 28. La imagen mostrada en una ventana de figura.

A continuación vemos el mensaje que nos muestra *MATLAB*, indicando que la imagen sale reducida de tamaño, concretamente un 50%:

```
Warning: Image is too big to fit on screen; displaying at 50%

> In imuitools\private\initSize at 72

  In imshow at 283
```

Esto tiene dos inconvenientes, por una parte muestra elementos que no sólo no son útiles sino que además pueden interferir en la captura de puntos, y por otra parte la imagen de cierta resolución sale comprimida y de un tamaño inferior para que pueda caber en la ventana. Como esto no se puede modificar debido a que está por defecto en el propio *MATLAB*, se tienen que buscar programas alternativos para mostrar las imágenes. En la base de datos de ficheros de intercambio de *MATLAB* hay un pequeño programa llamado *fullscreen*.

[*Web oficial*: http://www.mathworks.es/matlabcentral/fileexchange]

Este programa que se puede encontrar escribiendo *fullscreen* en la búsqueda de la página del link anterior esta creado por un usuario. Este pequeño paquete de archivos consiste en dos ficheros de extensión *m*, uno para mostrar la imagen en pantalla completa (*fullscreen.m*) y otro para cerrar la pantalla completa (*closescreen.m*).

```
%   FULLSCREEN(C,N) displays 24bit UINT8 RGB matlab image matrix C on
%   display number N (which ranges from 1 to number of screens). Image
%   matrix C must be the exact resolution of the output screen since
%   no scaling in implemented. If fullscreen is activated on the same
%   display as the MATLAB window, use ALT-TAB to switch back.
%
%   If FULLSCREEN(C,N) is called the second time, the screen will
%   update with the new image.
%
%   Use CLOSESCREEN() to exit fullscreen.
%
%   Requires Matlab 7.x (uses Java Virtual Machine), and has been
%   tested on Linux and Windows platforms.
%
%   Written by Pithawat Vachiramon 18/5/2006
```

El programa *fullscreen*, debido a que no tiene factores de escala implementados, requiere que el tamaño de la imagen sea el mismo de la pantalla donde se visualiza, también requiere que se esté trabajando con la versión de *MATLAB* 7.x ya que a partir de esta todas usan la *Máquina Virtual de Java* (*JVM*) y es gracias a *Java* que podemos efectuar la pantalla completa.

Debido a que las imágenes del patrón eran de clase doble y en escala de gris se ha modificado el programa ya que estaba diseñado para imágenes de clase entero de 8-bit y con canales de color RGB.

Otros dos programas usados frecuentemente para visualizar o editar los puntos son dos visualizadores de mallas y nubes de puntos: *MeshLab* y *CloudCompare*. Ambos son de código abierto y gratuito. Permiten abrir y modificar varios formatos de puntos, mallas y figuras.

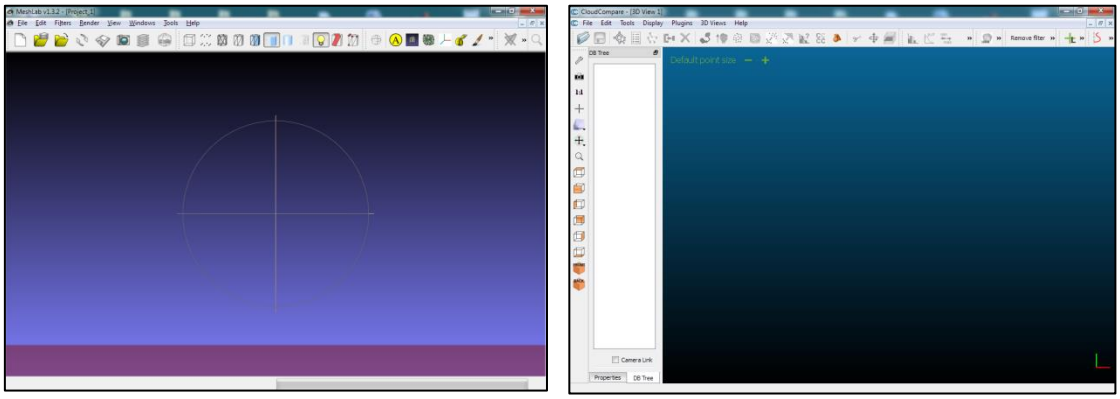


Fig. 29. *MeshLab* y *CloudCompare* respectivamente.

2.4.2 Metodología

Una vez expuesta toda la teoría e instrumentación empleada, a continuación se expone el método para hallar la nube de puntos del modelo con los siguientes pasos:

1- Calibración de las cámaras.
2- Proyección de la malla de puntos homólogos, patrón y captura de estos de forma simultánea con las dos cámaras.
3- Corregir las imágenes capturadas de distorsión.
4- Hallar las coordenadas de los puntos homólogos.
5- Hallar las coordenadas de los puntos que forman el patrón de franja.
6- Calcular los parámetros de orientación relativa del conjunto de cámaras.
7- Hallar la correspondencia entre cámaras.
8- Obtener las coordenadas modelo.

2.4.2.1 Patrón de luz

Una parte fundamental de este proyecto es la luz, tanto la condición de poca iluminación que es necesaria para poder efectuar un escaneado adecuado como el tipo de luz y su forma o diseño.

Concretamente se han empleado dos estructuras de luz. Por una parte tenemos una proyección blanca con una franja negra vertical que se va desplazando cada cinco píxeles hasta terminar con una imagen totalmente negra.

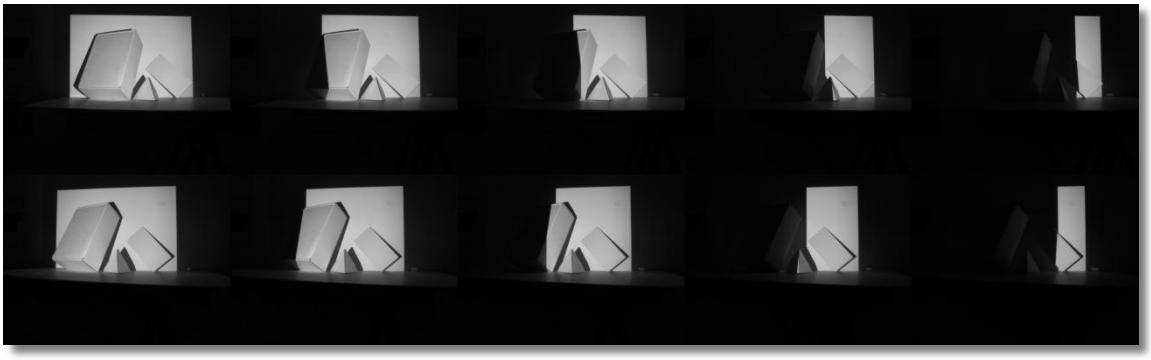
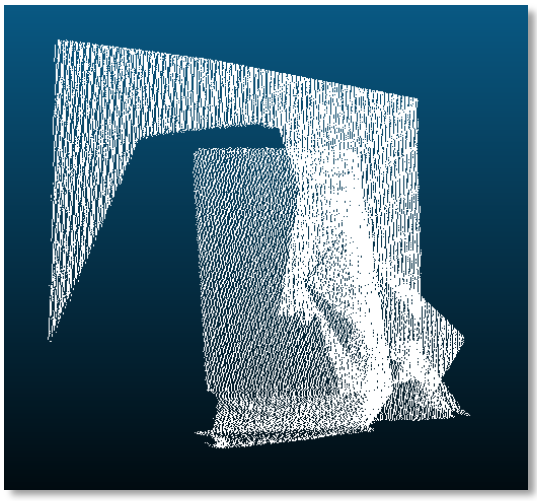


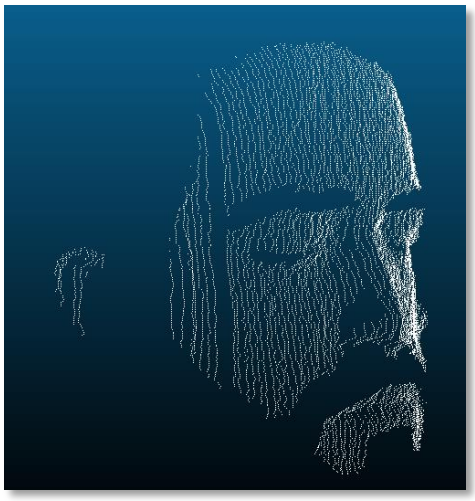
Fig. 30. Barrido de unas figuras con el patrón Blanco con franja Negra . El conjunto de arriba corresponde a la cámara izquierda y el de abajo a la derecha

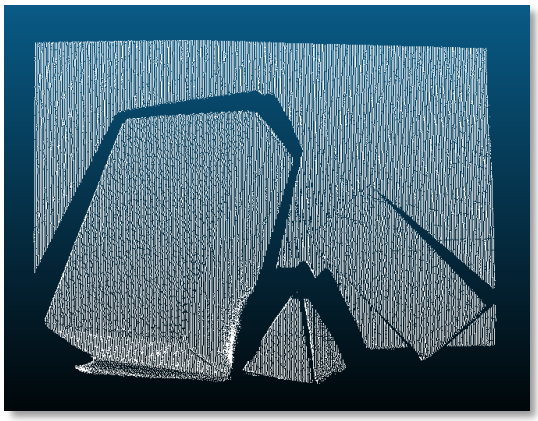
El otro patrón de luz consiste en una imagen totalmente negra con una franja blanca vertical con un ancho de cinco píxeles al igual que el otro patrón también se va desplazando por la pantalla cada cinco píxeles.

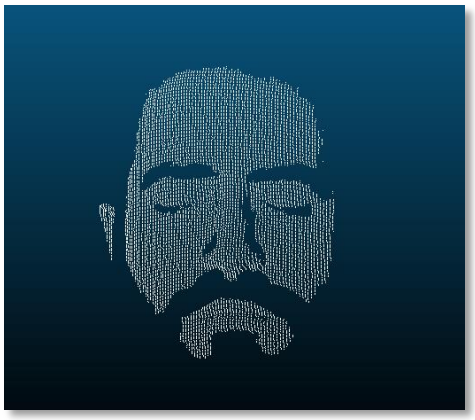
Fig. 31. Barrido de un rostro con el patrón de línea blanca. El conjunto de arriba corresponde a la cámara izquierda y el de abajo a la derecha.

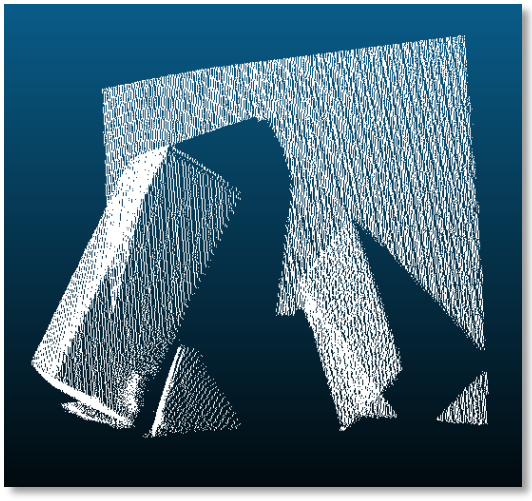
A continuación hay los dos conjuntos de figuras que se obtuvieron en estos dos casos siendo la de la izquierda el del primer patrón de luz y la derecha del segundo:











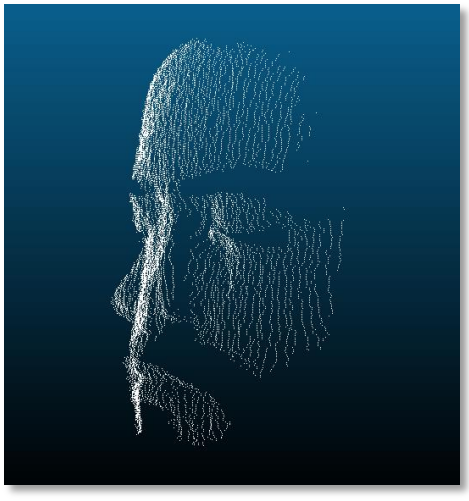


Fig. 32. Modelo de Figuras

Fig. 33. Modelo de cara

2.4.3 Paso a paso

Este apartado consiste en la explicación de forma detallada de cómo se lleva a cabo todo el proceso, partiendo con la calibración de cámaras y montaje del sistema desde la toma de imágenes hasta el modelo tridimensional de puntos. El método de obtención de las coordenadas de los puntos homólogos esta explicada previamente en el apartado 2.2.2 Puntos homólogos.

2.4.3.1 Captura de imágenes

Una vez el sistema está montado, anotamos la distancia a la que se sitúan las cámaras, en este caso siempre están separadas aproximadamente unos 270mm. Después de conectar las cámaras y el pico proyector al ordenador se abre *MATLAB* y el programa de captura de imagen.

Lo primero que hacemos es identificar cada cámara y adjudicarle un nombre para que el programa pueda llamar a la cámara y operar con ella. En este caso la cámara de la izquierda será *obj* y la cámara de la derecha será *obj2*. Como queremos modificar los parámetros de la cámara ya que por defecto la calidad de la toma fotográfica sale mal debido al tipo de luz ambiental, debemos definir el código fuente o *source*, de modo que cuando llamemos *src.Gamma* y le damos un valor específico estará modificando en directo el parámetro gamma de la cámara.

```
%Identificamos las dos cámaras y sus propiedades para poder
modificarlas.
obj = videoinput('winvideo', 2);
obj2= videoinput('winvideo', 3);
src = getselectedsource(obj);
src2= getselectedsource(obj2);
%Modificamos gain, gamma y exposición de las dos cámaras.
src.Gain=400;
src.Gamma=110;
src.Exposure = -4;
src2.Gain=400;
src2.Gamma=110;
src2.Exposure = -4;
```

Debido a que el programa utilizado para mostrar las imágenes en pantalla completa desde Matlab requiere que el tamaño de imagen sea el mismo que el de la pantalla, antes de empezar a mostrar y capturar se debe recoger la información del tamaño de imagen y preparar las variables que definirán el tamaño del patrón.

```
screen_size = get(0, 'ScreenSize');
N=screen_size(4);
M=screen_size (3);
```

Para reducir al máximo el tiempo de escaneo, las imágenes del patrón se crean directamente dentro del bucle y se guardan en la memoria del sistema como una matriz de *N* dimensiones, siendo *N* el número de fotogramas capturados.

```
for i=1:5:M-5
    contador=contador+1;
    img=zeros(N,M);
    img(:, i:i+5)=1;
    fullscreen(img,1);

    %Toma de imagen
    frame(:,:,contador) = getsnapshot(obj);
    frame2(:,:,contador)= getsnapshot(obj2);

    pause(0.1)
end
closescreen;
```

Finalmente cuando se han tomados todos los fotogramas del patrón se almacenan en la memoria del disco duro. De esta forma se reduce el tiempo de escaneo hasta 60 segundos.

```
for i=1:contador
    cd(folder_name);
    file_name1 = [ 'izq_00' num2str(i) '.bmp'];
    imwrite(frame(:,:,i),file_name1);

    file_name2 = [ 'der_00' num2str(i) '.bmp'];
    imwrite(frame2(:,:,i),file_name2);
end
```

A continuación muestro una captura de pantalla del tiempo que tarda el programa en hacer el escaneo total. Este proceso, que actualmente tarda 231 segundos, se podría recortar significativamente si se trabajara con video, pero para ello sería necesario unas cámaras que pudieran capturar video con una resolución más alta.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
capturaDirectePROVAUNO	1	231.002 s	37.998 s	

Fig. 34. Tiempo que tarda el proceso

2.4.3.2 Corrección de distorsión

Como se ha comentado previamente es muy importante corregir la distorsión de las lentes pues nos puede dar unos resultados finales erróneos. Para ello se ha creado un programa donde primero hay que introducir un directorio donde se sitúan las imágenes tomadas con la cámara derecha e izquierda y los parámetros internos de ambas cámaras:

```
x0 = 2.979974 ;
y0 = 1.937599;
k1= (-6.453995e-003);
k2= (1.470374e-004);
p1= (-2.052425e-005);
p2= (-3.218858e-005);
```

A continuación se abre la imagen, y píxel por píxel aplica los cambios corrigiendo la distorsión:

```
for i=1:columna
      for j=1:fila
            x= i*0.0045;
            y= j*0.0045;
            r = ((x-x0)^2+(y-y0)^2)^0.5;
            rp = (k1*r^3+k2*r^5);
            xx =(x-x0)+(x-x0)*rp/r + p1*(r^2+2*(x-x0)*...
                (x-x0))+2*p2*(x-x0)*(y-y0);
            yy =-((y-y0)+(y-y0)*rp/r + p2*(r^2+2*(y-y0)*...
                (y-y0))+2*p1*(x-x0)*(y-y0));
            ip=fix(xx/0.0045+1280/2);
            jp=fix(960-(yy/0.0045+960/2));
```

Finalmente sitúa el píxel en su nueva posición correcta, aplicando siempre la condición para que el punto sólo sea incluido si está situado dentro del tamaño de la imagen, esto es debido a que a veces al corregir distorsiones hay puntos en los límites que salen fuera del marco de la imagen original:

```
if (jp>=1) && (jp<=fila) && (ip>=1) && (ip<=columna)
      izqCorr(j,i) = izq(jp,ip);
end
```

A las imágenes nuevas se les añade un sufijo al nombre para saber que están corregidas:

```
[pathstr,name,ext] = fileparts(filename);
filename1 = [name, '_CI', '.bmp'];
imwrite(ZIzq,filename1);
```

Este proceso se repite para todas las imágenes en el directorio y luego para la otra cámara con sus parámetros de distorsión correspondientes.

A continuación vemos un ejemplo de una toma fotográfica, por una de las cámaras, a un tablero mostrado en la pantalla del ordenador. Veremos la imagen sin corregir con una clara distorsión en barrilete, la misma imagen corregida y finalmente una sobre posición de ambas imágenes centradas:

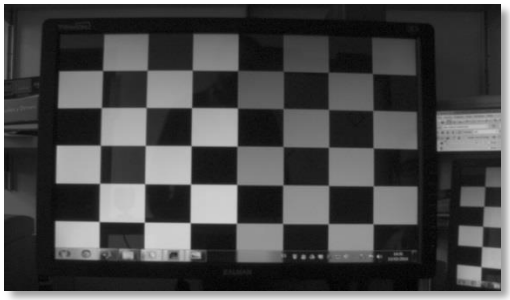


Fig. 35. Imagen del tablero sin corregir

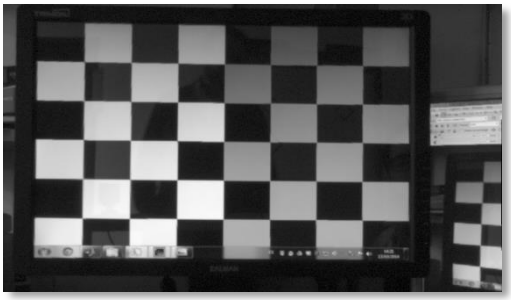


Fig. 36. Imagen del tablero corregida

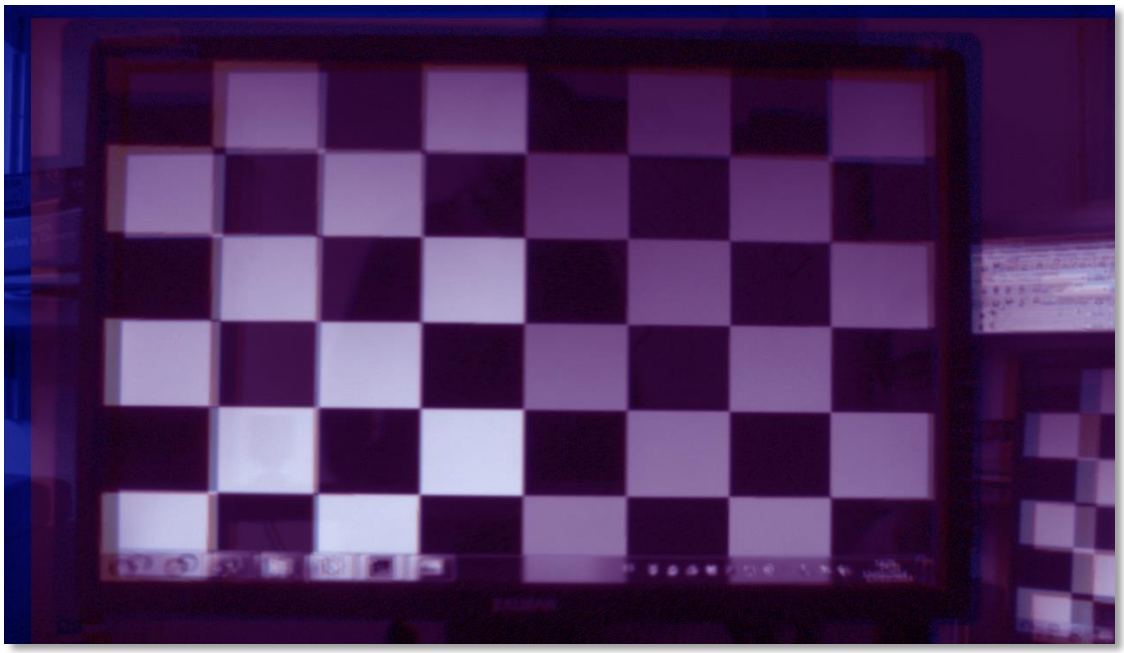


Fig. 37. Imágenes del tablero sin y con corrección superpuestas

2.4.3.3 Coordenadas del patrón de luz

Este apartado consiste en la explicación de cómo se consiguen las coordenadas de las franjas formadas por el patrón de luz. Concretamente para hallar la coordenada se define un rango de valor de intensidad de píxel. Una vez se ha definido dónde están las imágenes y cuáles son, he definido un bucle que escanea cada columna de cada fila, es decir píxel por píxel, hasta hallar uno que este dentro del rango de intensidad.

```
if (I(j,i)>=90) && (I(j,i)<=250)
```

Lógicamente este método está muy limitado por la luz ambiental, donde un caso ideal sería una habitación totalmente oscura.

Una vez el programa ha encontrado el píxel que corresponde a la franja lo guarda. Debido a que la imagen está representada por píxeles, las coordenadas siempre las encontrara en números enteros, esto causa un problema al representar los modelos ya que los puntos quedan agrupados y dan una sensación de escalonado, esto se puede ver en las siguientes figuras:

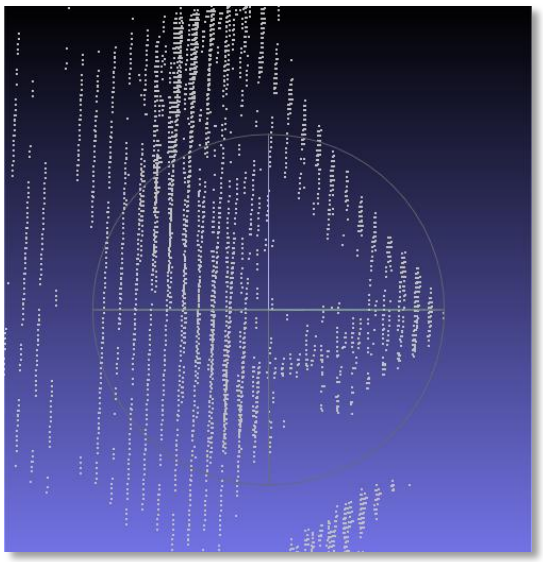


Fig. 38. Perfil de nariz con escalonado

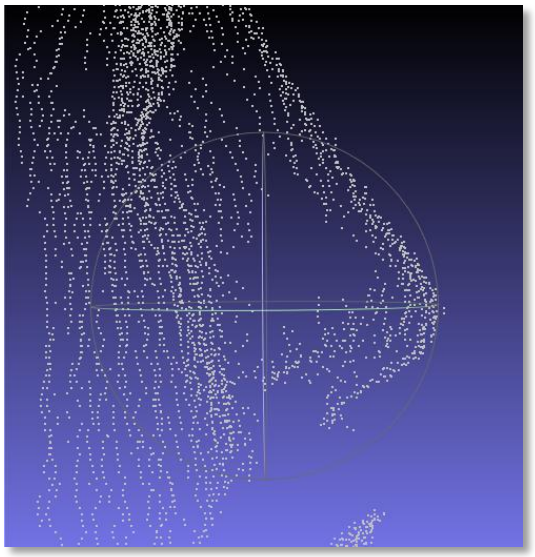


Fig. 39. Perfil de nariz con promedio.

Para evitar esto una vez el programa encuentra el píxel de la franja hace un promedio de los cinco píxeles previos y posteriores de la misma fila de la siguiente manera:

$$x' = \frac{\sum ND \cdot x}{\sum ND}$$

Siendo:

-*ND* el nivel digital o valor de gris de cada píxel.

-*x* siendo la coordenada o en este caso la posición de columna.

Este promedio no sólo nos da el valor de *x* en decimal sino que además nos ajusta más la franja en el caso del patrón de línea blanca.

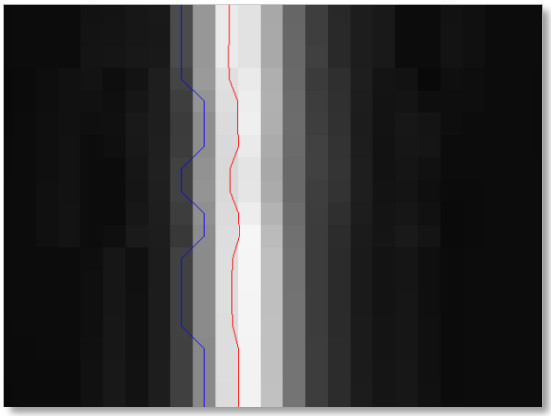


Fig. 40. Línea azul es la escalonada y la línea roja es la promedio.

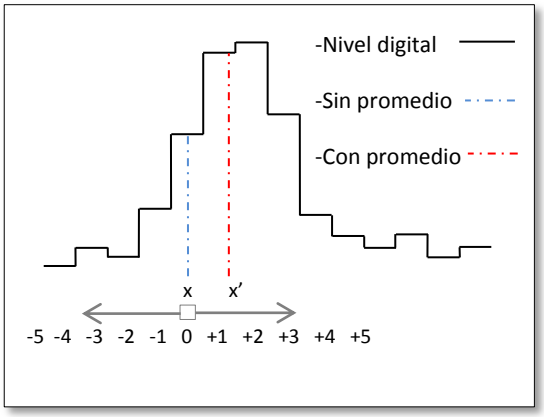


Fig. 41. Ejemplo teórico del promedio

Ejemplo de la selección de píxel y promediado del mismo:

```
if (I(j,i)>=90) && (I(j,i)<=250) % condición de nivel de gris
      %Paquete de 5 posiciones a la izquierda y derecha
      A=double([i-5,i-4,i-3,i-2,i-1,i,i+1,i+2,i+3,i+4,i+5]);
      %Nivel de gris de cada posición
      gris=double([ I(j,A(1)), I(j,A(2)), I(j,A(3)), I(j,A(4)),...
            I(j,A(5)),I(j,A(6)),I(j,A(7)),I(j,A(8)),I(j,A(9)), ...
            I(j,A(10)),I(j,A(11))]);

      grisX=gris.*A; % ND*x
      X=(sum(grisX)/sum(gris)); %Promedio
      PX(j,:)=[X j]; %Almacenar coordenada en matriz de puntos
```

Este proceso se lleva a cabo para todos los píxeles de cada imagen de ambas cámaras. Finalmente obtendremos un fichero texto con dos columnas y *N* filas para cada franja, es decir cada imagen.

2.4.3.4 Orientación relativa

La orientación relativa es una de las partes más laboriosas de implementar en el código ya que hay que escribir muchas funciones y matrices.

Para ello se deben definir los parámetros iniciales. También es necesario especificar que para la primera iteración los parámetros de la matriz de rotación serán todos cero.

```
%Parámetros iniciales
bx=270;
by=0;
bz=0;
%f1=izquierda, f2= derecha
f1=5.993458;
f2=5.981018;
omeI=0;

%Los parámetros que van modificándose con cada iteración.
phiI=x(1);
kapI=x(2);
omeD=x(3);
phiD=x(4);
kapD=x(5);
```

Tras preparar los parámetros iniciales hay que cargar los puntos homólogos calculados previamente de forma que se ponen en una matriz de tres columnas donde la primera y segunda columna son las coordenadas *x* e *y* respectivamente y la tercera columna es la focal de la cámara correspondiente.

```
%Montamos dos matrices con los puntos izquierdos y derechos.
for e=1:L
    PIz(e,:)=[puntos(e,1),puntos(e,2),-f1];
end

for e=1:L
    PDe(e,:)=[puntos(e,3),puntos(e,4),-f2];
End

P= [PIz,PDe];
```

Definimos las matrices de rotación y creamos la matriz de diseño:

```
RI=[cos(phiI)*cos(kapI), cos(phiI)*sin(kapI),-sin(phiI);
    -cos(omeI)*sin(kapI)+sin(omeI)*sin(phiI)*cos(kapI), ...
    cos(omeI)*cos(kapI)+sin(omeI)*sin(phiI)*sin(kapI),...
    sin(omeI)*cos(phiI);
    sin(omeI)*sin(kapI)+cos(omeI)*sin(phiI)*cos(kapI),...
    -sin(omeI)*cos(kapI)+cos(omeI)*sin(phiI)*sin(kapI) , ....
    cos(omeI)*cos(phiI)];

RD=[cos(phiD)*cos(kapD), cos(phiD)*sin(kapD),-sin(phiD);
    -cos(omeD)*sin(kapD)+sin(omeD)*sin(phiD)*cos(kapD),...
    cos(omeD)*cos(kapD)+sin(omeD)*sin(phiD)*sin(kapD),...
    sin(omeD)*cos(phiD) ;
    sin(omeD)*sin(kapD)+cos(omeD)*sin(phiD)*cos(kapD),....
    -sin(omeD)*cos(kapD)+cos(omeD)*sin(phiD)*sin(kapD), ....
    cos(omeD)*cos(phiD)];

RI=RI';
RD=RD';

VI= RI(2,1)*P(i,1)+RI(2,2)*P(i,2)+RI(2,3)*P(i,3);
WI=RI(3,1)*P(i,1)+RI(3,2)*P(i,2)+RI(3,3)*P(i,3);

VD= RD(2,1)*P(i,4)+RD(2,2)*P(i,5)+RD(2,3)*P(i,6);
WD= RD(3,1)*P(i,4)+RD(3,2)*P(i,5)+RD(3,3)*P(i,6);

%Creamos la matriz de diseño A.
A11= -( P(i,1)*sin(phiI)+ f1*cos(phiI) )*sin(kapI)*WD +...
    VD*(P(i,1)*cos(phiI)- f1*sin(phiI));
A12=  ( P(i,1)*cos(phiI)*cos(kapI) - P(i,2)*sin(kapI) - ...
    f1*sin(phiI)*cos(kapI))*WD;
A13= ( P(i,5)*cos(omeD)*cos(phiD)+f2*sin(omeD)*cos(phiD))*VI...
    -(-P(i,5)*sin(omeD)*cos(kapD)+P(i,5)*cos(omeD)*sin(phiD)...
    *sin(kapD)+f2*(cos(omeD)*cos(kapD)+sin(omeD)*sin(phiD)...
    *sin(kapD) ))*WI;
A14= (-P(i,4)*cos(phiD)-P(i,5)*sin(omeD)*sin(phiD)+f2*cos(omeD)...
    *sin(phiD))*VI + (P(i,4)*sin(phiD)*sin(kapD)-P(i,5)*sin(omeD)...
    *cos(phiD)*sin(kapD)+f2*cos(omeD)*cos(phiD)*sin(kapD))*WI;
A15=-( P(i,4)*cos(phiD)*cos(kapD)-P(i,5)*cos(omeD)*sin(kapD)...
    +P(i,5)*sin(omeD)*sin(phiD)*cos(kapD)-f2*cos(omeD)*sin(phiD)...
    *cos(kapD)-f2*sin(omeD)*sin(kapD) )*WI;


A(i,:)=[A11,A12,A13,A14,A15];
U(i)= -(VI*WD-VD*WI);
```

Seguidamente aplicamos mínimos cuadrados y preparamos los parámetros para la siguiente iteración añadiendo una condición al final del código.

```
%Mínimos cuadrados
dx=A\U';
V=A*dx-U';

x=x+dx;
condition=0;

%La condición para que el programa pare de hacer iteraciones
    for j=1:5
        if abs(dx(j))>= 1e-6
            condition=1;
        end
    end
```

Finalmente obtendremos los parámetros de orientación relativa de ambas cámaras, estos nos permitirá hallar la correspondencia entre imágenes.

```
RI =

   0.9963   -0.0084   -0.0851

   0.0083    1.0000   -0.0007

   0.0851         0    0.9964
```

```
RD =

    0.9994   -0.0079    0.0338

    0.0084    0.9999   -0.0138

   -0.0337    0.0140    0.9993
```

2.4.3.5 Correspondencia entre imágenes

Una vez hemos obtenido los parámetros de orientación, es decir las matrices de rotación y traslación de ambas cámaras, es hora de epipolarizar los puntos de las franjas mediante las ecuaciones de colinealidad.

Ante todo, lo primero que haremos será cargar los archivos que contienen las coordenadas de las franjas del patrón izquierdo y derecho. Una vez cargados crearemos dos matrices una para la izquierda y derecha transformando las coordenadas de píxel a mm y refiriéndolas al punto principal de la cámara.

```
lp=load(filename);
rp=load(filename2);

for i=1:LL
left(i,:)=[(lp(i,1)*0.0045-2.979974) (lp(i,2)*0.0045-(-1.937599))];
end

for i=1:RR
right(i,:)=[(rp(i,1)*0.0045-2.725877) (rp(i,2)*0.0045-(-2.093914))];
end
```

Aplicamos las ecuaciones de colinealidad:

```
for i=1:LL
    xpi(i) = -f1*(left(i,1)*RI(1,1) + left(i,2)*RI(1,2)...
        - f1*RI(1,3))/(left(i,1)*RI(3,1) + left(i,2)*RI(3,2)...
        - f1*RI(3,3));
    ypi(i) = -f1*(left(i,1)*RI(2,1) + left(i,2)*RI(2,2) ...
        - f1*RI(2,3))/(left(i,1)*RI(3,1) + left(i,2)*RI(3,2)...
        - f1*RI(3,3));
end

for i=1:RR
     xpd(i) = -f2*(right(i,1)*RD(1,1) + right(i,2)*RD(1,2)...
         - f2*RD(1,3))/(right(i,1)*RD(3,1) + right(i,2)*RD(3,2)...
         - f2*RD(3,3));
     ypd(i) = -f2*(right(i,1)*RD(2,1) + right(i,2)*RD(2,2)...
         - f2*RD(2,3))/(right(i,1)*RD(3,1) + right(i,2)*RD(3,2)...
         - f2*RD(3,3));
end
```

Finalmente hallamos la correspondencia entre las coordenadas de la franja izquierda y derecha buscando una coordenada en la derecha dentro de un intervalo de la coordenada de la izquierda.

```
for i=1:LL
    minfrizq= frizq(i,2)-3e-3;
    maxfrizq= frizq(i,2)+3e-3;
    for j=1:RR
        if (frder(j,2)>=minfrizq) && (frder(j,2)<=maxfrizq)
            AAA(i,:)=[frizq(i,:) frder(j,:)];
        end
    end
end
```

2.4.3.6 Obtención de coordenadas modelo

La parte final del procedimiento consiste en hallar las coordenadas del modelo, para ello aplicaremos los factores de escala *M* y *N* en el siguiente código:

```
for g=1:G
    denominador=((AAA(g,1)*-f2)-(-f1*AAA(g,3)));
    if abs(denominador)>= 1e-6
        M=-f1*bx/denominador;
        N=-f2*bx/denominador;
        x1=AAA(g,1)*N;
        x2=AAA(g,3)*M+bx;
        y1=AAA(g,2)*N;
        y2=AAA(g,4)*M;
        z=-f1*N;
        XYZ(g,:)= [x1 (y1+y2)/2 z];
    end
end
```

Repetimos el proceso para todos los pares de coordenadas (izquierda y derecha) y obtendremos una nube de puntos de la escena u objeto escaneado.

2.4.4 Cálculo de errores

Como con todo trabajo técnico, hay que comprobar si los resultados que obtenemos son válidos. En este caso una forma de ver si el trabajo tenía buenos resultados era escaneando objetos planos, como una pared, caja o cartulina.

En este caso tras escanear y obtener la nube de puntos de unas figuras, con el programa *Meshlab* se eliminaron los puntos que no interesaban hasta tener solamente los que formaban uno de los lados de la caja:

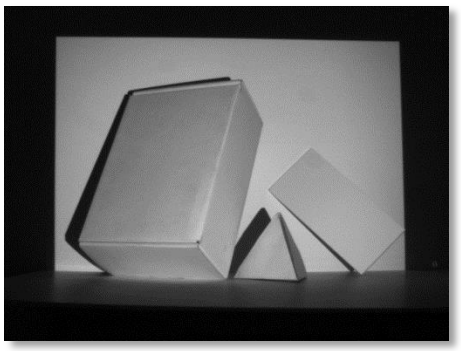


Fig. 42. Toma fotográfica de las figuras.

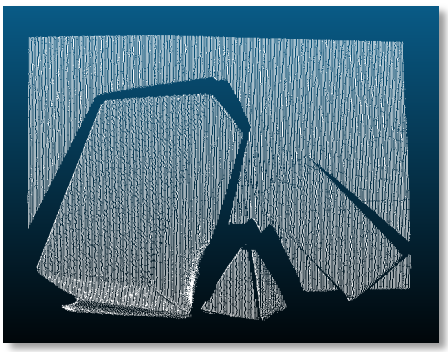


Fig. 43. Nube de puntos de las figuras.

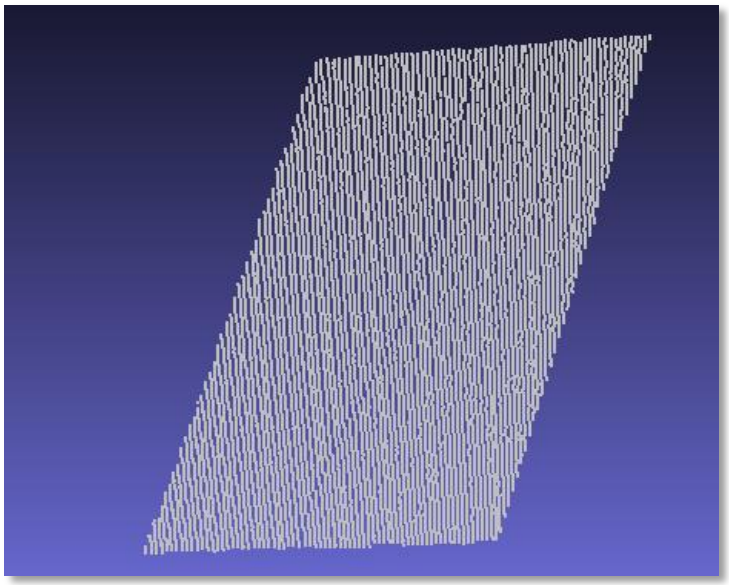


Fig. 44. Nube de puntos de un lado de la caja

Tras pasar estos puntos por un programa que determina la distancia punto-plano obtenemos la distancia que se aleja cada punto del plano formado por el conjunto de puntos.

```
P= load('PlanoCajaCorregido.txt');

[M, N]= size(P);
A= zeros(M,3);
U= zeros(M,1);
 for i=1:M
    A(i,:) = [P(i,1) P(i,2) P(i,3)];
    U(i,:)= -1;
 end

x = A\U; %Mínimos cuadrados
for i=1:M
    D(i,:)= (x(1)*P(i,1)+x(2)*P(i,2)+x(3)*P(i,3)+1)...
        /(x(1)*x(1)+x(2)*x(2)+x(3)*x(3))^0.5;
end

EMC= ((D'*D)/(M-1))^0.5;

XYZD= [P D ];
```

Si pedimos a *MATLAB* que nos dé el máximo y mínimo de los valores de la distancia en milímetros:

```
>> max(D)

ans = 2.4150

>> min(D)

ans = -1.8923
```

Es decir que la distancia máxima que un punto se aleja del plano es de 2.415mm.

Pedimos la media y la mediana:

```
>> mean(D)

ans =8.1147e-04

>> mean(abs(D))

ans =0.4911

>> median(abs(D))

ans =0.4237
```

Si hacemos lo mismo con el plano que forma la cartulina pequeña de la misma nube de puntos:

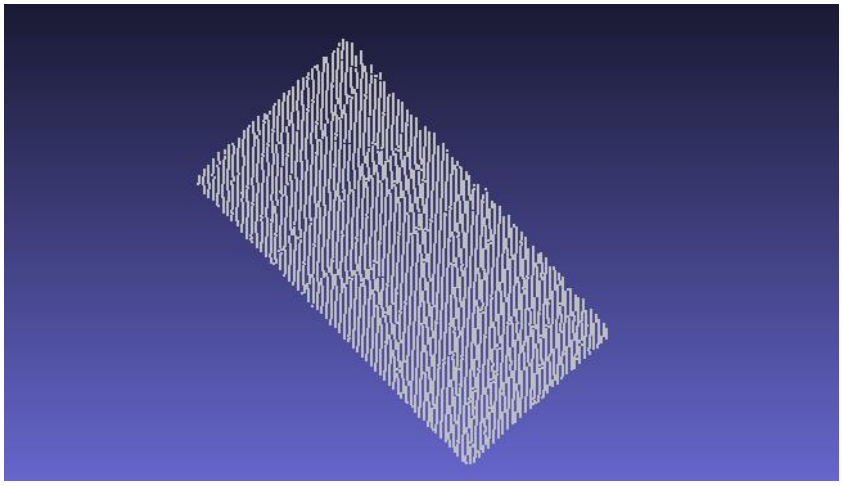


Fig. 45. Nube de puntos de la cartulina pequeña

Tras pasar los puntos por el programa *determinación de planos* obtendremos otra vez un listado de distancias punto-plano. Si volvemos a pedir a *MATLAB* que nos calcule los valores máximos, mínimos, mediana y media:

```
>> max(D)

ans = 2.1485

>> min(D)

ans = -2.5641

>> mean(D)

ans =7.6142-04

>> mean(abs(D))

ans =0.5688

>> median(abs(D))

ans =0.4843
```

Con estos resultados podemos concluir que el sistema es efectivo y somos capaces de apreciar aproximadamente entre 1/4 y 1/5 del tamaño de píxel.

También se ha llevado a cabo un experimento escaneando un mismo conjunto de objetos mediante nuestro sistema y un digitalizador 3D *Vivid 910* de la casa *Konica Minolta.* (Fig.)

Este instrumento permite, de forma semejante al sistema creado en este proyecto, encontrar una nube de puntos de un objeto a un distancia de entre 0.6 y 2.5m. A diferencia del equipo de este trabajo, este sistema proyecta una línea láser rojo, pero también hace un barrido y este mismo es capturado por una lente situada al otro extremo del aparato.

La precisión con la que trabaja entre 8 y 32 µm, dependiendo de cuál de las tres lentes lleva puesta.

Fig. 46. Vivid 910

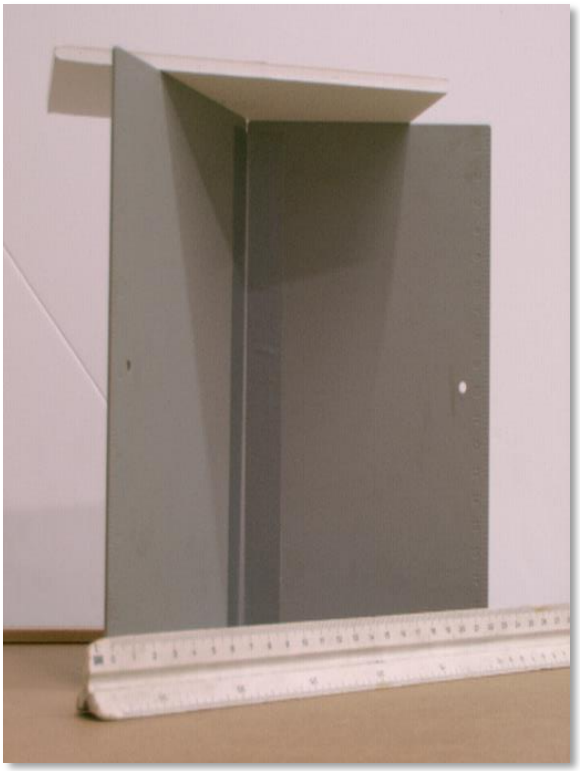


Fig. 47. Conjunto de objetos escaneados

A continuación vemos los objetos de la figura 47 escaneados por los dos sistemas, debido al sistema de montaje de ambos, las tomas están hechas desde puntos de vista diferentes:

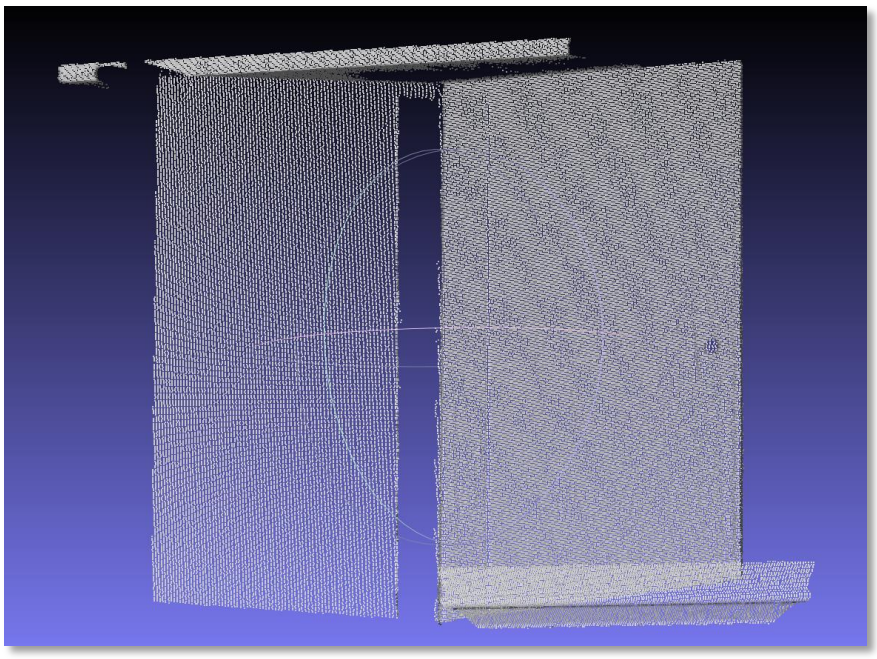


Fig. 48. Nube de puntos con Vivid 910

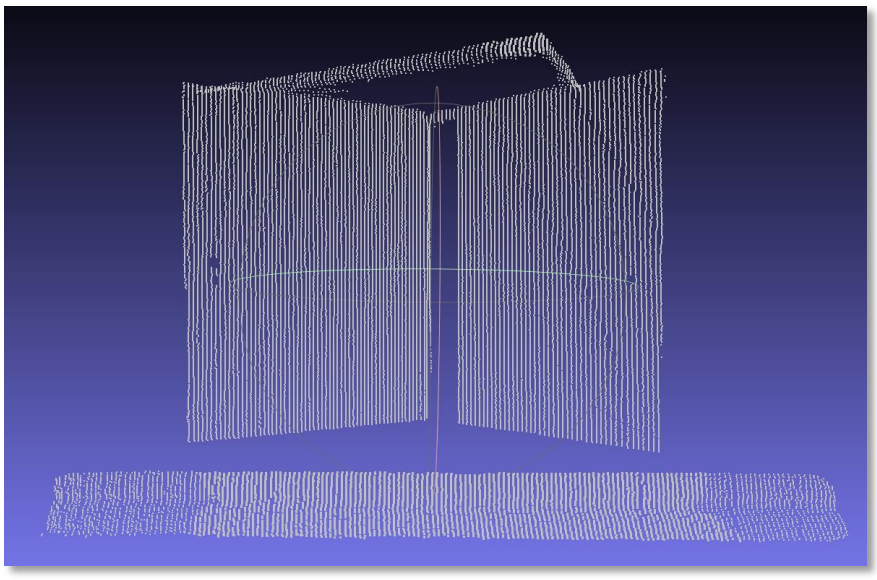


Fig. 49. Nube de puntos con el sistema propio.

Otro ejemplo de digitalización, esta vez con un rostro:

Fig. 50. Rostro a escanear

Fig. 51. Nube de puntos con Vivid 910

Fig. 52. Nube de puntos del sistema propio.

En este caso del rostro los escaneos no se efectuaron el mismo día, por eso hay diferencias en las figuras 51 y 52 con el vello facial y la ropa, pero se puede apreciar que lógicamente el sistema *Vivid 910* captura unas nubes de puntos mucho más densas, de hecho la imagen parece tener textura de sólido debido a la cantidad de puntos.

También hay que decir que si en el sistema propuesto en este trabajo se le añadieran cámaras de mayor resolución y modificando el ancho de barrido se podría obtener un resultado muy similar al sistema de *Konica Minolta* sin tener que recurrir a una proyección láser.

A continuación se muestran pequeñas zonas de interés donde se puede apreciar la diferencia de densidad de puntos entre ambos sistemas:

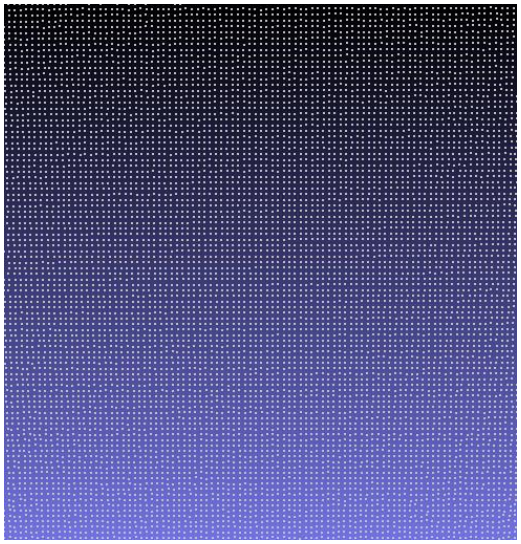


Fig. 53. Región plano con Vivid 910

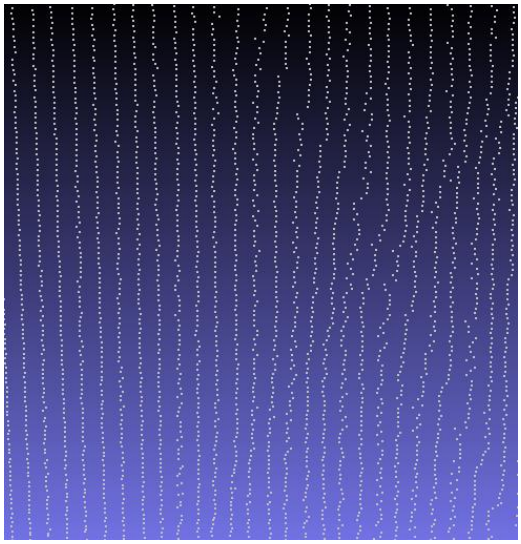


Fig. 54. Región plano con el sistema propio

Si comparamos los resultados estadísticos de la distancia punto-plano de los planos extraídos de las figuras 53 y 54 respectivamente:

```
max(D)
ans =1.9077
>> min(D)
ans = -0.5460
>> mean(D)
ans = 2.7961e-05
>> mean(abs(D))
ans = 0.1101
>> median(abs(D))
ans =0.0912
```

```
>> max(D)
ans = 1.9589
>> min(D)
ans =-1.8784
mean(D)
ans = 5.0891e-04
>> mean(abs(D))
ans =0.5281
>> median(abs(D))
ans = 0.4766
```

Si comparamos resultados, como era de esperar los valores estadísticos del *Vivid 910* son mejores pero el valor de la distancia máxima punto-plano es muy parecido.

3 CONCLUSIONES

1. Para poder hacer un método de escaneado más rápido se han tenido que prescindir de unas cámaras de mayor resolución, pasando de una hora y treinta minutos a aproximadamente cuatro minutos de tiempo de captura.

2. Se ha diseñado un sistema pequeño y ligero con dos cámaras *DMK 41BU02* de la marca *ImagingSource* con un proyector *Acer* modelo *c120* que permiten dar movilidad al usuario pudiendo estacionar libremente.

3. La extensión *IC Matlab Plugin for Matlab* junto con el propio *Matlab* han sido vitales para poder garantizar una sincronización y rapidez en la toma fotográfica de los modelos.

4. De los dos patrones de luz utilizados, el barrido de blanco al negro da resultados erróneos con las sombras dándoles profundidad en algunos casos, en el barrido de línea blanca no se da este caso.

5. Los programas desarrollados en este trabajo permiten:

- Proyección y captura de patrones de luz.
- Hallar las coordenadas del centro y radios de puntos homólogos.
- Corregir imágenes de distorsión radial y tangencial.
- Medir fotocoordenadas en el par de imágenes.
- Calcular los parámetros de orientación relativa entre dos cámaras.
- Epipolarizar imágenes.
- Obtener coordenadas modelo de un objeto mediante sucesivas fotografías.
- Determinar planos y distancias punto-plano.

6. Debido a la naturaleza del sistema de proyección y captura de imagen este sistema está muy limitado a la luz ambiental. Para un caso ideal de trabajo sería necesario un nivel de luz ambiental muy bajo. Esto se podría solucionar mediante un proyector de luz infrarroja y una cámara capaz de capturar este tipo de luz.

7. En caso de disponer de cámaras de alta resolución y modificando el ancho de barrido a 1 píxel de podría aumentar el detalle y la precisión en los modelos significativamente.

4 BIBLIOGRAFIA

Libros

Buill, Felipe; Mª Amparo Núñez; Joan Rodríguez Jordana. *Fotogrametría analítica.* Ed. UPC (2003)

Faugeras, Olivier; Quang-Tuan Luong. *The geometry of multiple images*. Ed. M.I.T (2001)

Gonzalez, Rafael C; Richard E. Woods. *Tratamiento digital de imágenes*. Ed. Díaz de Santos (1996)

Gonzalez, Rafael C; Richard E. Woods; Steven L. Eddins. *Digital Image Processing Using MATLAB.* Ed. Gatesmark Publishing (2009)

Lerma García, J. L. *Fotogrametría moderna: analítica y digital.* Ed. UPV (2002)

Lerma García, J.L. *Problemas de fotogrametría II*. Ed. UPV (1999)

Páginas Web

http://www.mathworks.es/es/help/vision/

http://www.mathworks.es/matlabcentral/fileexchange

http://www.theimagingsource.com/en_US/

http://www.vision.caltech.edu/bouguetj/

http://www.getreuer.info/tutorials/matlabimaging

AGRADECIMIENTOS

El proyecto final de carrera no hubiera podido llevarse a cabo sin la ayuda y orientación de mis tutores, Felipe Buill por su infinita paciencia y Albert Prades por sus brillantes intervenciones en momentos donde el proceso de programación parecía no avanzar.

Estoy especialmente agradecido también a mi familia por su paciencia y apoyo incondicional y económico durante toda la carrera, sobre todo en esta recta final.